FingerCell 2.1 EDK

FingerCell 2.1 EDK

Published July 25, 2007. Version 2.1.0.0

Table of Contents

List of Tables

Chapter 1. About

1.1. Introduction

Before using FingerCell EDK components protection service must be started.

- ARM Linux users should start /bin/linux_arm/activation/run_pgd.sh
- Windows Mobile users using ipag 5500 series should start \bin\ppc03_ipaq\activation\pg.exe
- Windows Mobile (MS Windows CE 2003 or later) users should start \bin\ppc03_armv4\activation\pg.exe

Note for Windows Mobile users: Start program pg.exe and close its window, it will stay in background, this must be done each time you reboot your Windows Mobile)

FingerCell EDK consists of NFRecord, FncExtractor, FncMatcher main libraries. NCore and NImages are auxiliary libraries that provides infrastructure and functionality for working with images. The samples programs were developed to show how to use FingerCell EDK components.

1.2. Platforms Supported

FingerCell EDK supports platforms based on arm processor architecture. Libraries for Windows CE and Linux operating systems are provided.

1.3. Licensing

Neurotechnologija grants you a personal, non-exclusive license to use the Software for 30 consecutive days on one embedded device for the purpose of technology evaluation.

Chapter 2. What's New

2.1. Version 2.1.0.0

- Improved speed of fingerprint template extraction.
- New functionality in FncExtractor library enabling faster fingerprint template extraction for verification/identification with few templates scenarios.

2.2. Version 2.0.0.0

- Improved reliability of fingerprint template extraction and matching.
- Matching algorithm has now one speed and is faster and more reliable than in FingerCell 1.2.
- Added algorithm optimizations for more fingerprint scanners.
- New EDK structure.
- FncExtractor and FncMatcher libraries should be now used instead of FingerCell library to use all features of FingerCell algorithm.

2.3. Version 1.2.0.0

- Improved reliability.
- Better matching performance. Both matching speeds are now about 50% faster than in version 1.1.
- Reduced template size. Now template occupies 150 - 300 bytes (vs. 200 - 650 in version 1.1).
- Features compression and decompression functions are now built in the library.
- Added CrossMatch Verifier 300, DigitalPersona U.are.U, Atmel FingerChip, STMicroelectronics TouchChip, BMF BLP-100 and Secugen Hamster scanners' modes. Authentec sensors' modes are now separate from General mode.
- FingerCell can now return skeletonized image.

2.4. Version 1.1.0.0

- Completely new interface and a number of improvements in the algorithm.
- Higher matching speed. Now only two speeds are available - low (0 speed in 1.0) and high (5 speed in 1.0). Both speeds are faster that in version 1.0.
- Improved recognition reliability.
- Improved fingerprints extraction algorithm - now extraction can work with various image resolutions (1.0 version worked only with 250 dpi fingerprint images).

Chapter 3. Overview

3.1. Image Support

Image support in the FingerCell EDK can be divided into the following three parts:

- Image. The base of all image support. Developers should start using this part and take advantage of other parts if it is required.
- Image Format. Declares the supported image formats. Shows how to load and save images in a format-neutral way.
- Low-Level Image Input-Output. Should be used to have more control on how images are loaded and saved in particular format.

3.1.1. Image

Image is a rectangular area of pixels (image elements), defined by width, height and pixel format.

Pixel format describes type of color information contained in the image like monochrome, grayscale, true color or palette-based (indexed) and describes pixels storage in memory (how many bits are required to store one pixel).

Image in the FingerCell EDK is defined by HNImage handle in NImage module. It is an encapsulation of a memory block that stores image pixels. The memory block is organized as rows that follow each other in top-to-bottom order. The number of rows is equal to height of image. Each row is organized as pixels that follow each other in left-to-right order. The number of pixels in a row is equal to width of image. A pixel format describes how image pixels are stored. See `NImageGetWidth`, `NImageGetHeight`, `NImageGetStride`, `NImageGetPixelFormat` and `NImageGetPixels` functions for more information.

An image can have horizontal and vertical resolution attributes assigned to it if they are applicable (they are required for image, and do not make sense for face image). See `NImageGetHorzResolution` and `NImageGetVertResolution` functions for more information.

An image can be created either as empty or from existing memory block. See `NImageCreate`, `NImageCreateFromData` and `NImageCreateWrapper` functions for more information.

For each value of NPixelFormat exposed via interface a module is provided for managing according type of image (getting and setting individual pixels, etc.). See NGrayscaleImage, NMonochromeImage and NRgbImage modules for more information.

An image can be converted to different pixel format using `NImageCreateFromImage` or `NImageCreateFromImageEx` function.

Different methods should be used to display an image on different platforms:

- On Windows `BmpSaveImageToHBitmap` function can be used to receive a standard Win32 HBITMAP for the image. The reverse process is also possible using `BmpLoadImageFromHBitmap` function.
- On Linux there is no easy method implemented. However, a memory block containing pixels of image could be accessed via `NImageGetPixelFormat` function. The memory block can be used to display the image or convert it to some other representation on any platform.

An image can be stored in file in any supported image format using `NImageSaveToFile` function.

An image stored in file in any supported image format can be loaded using `NImageCreateFromFile` function.

3.1.2. Image Format

Image format is a specification of image storage in a file. The specification may require to compress/decompress image during writing/reading it to/from a file.

Image format in the FingerCell EDK is defined by HNImageFormat handle in NImageFormat module.

There is a number of image formats supported in the FingerCell EDK. Certain formats could not be read from and written to a file on all platforms. See the following table for details.

Image Format	**Can read**	**Can write**
BMP	Yes	Yes
GIF	No	No
JPEG	Yes	Yes
PNG	No	No
TIFF	Yes	No

These image formats are accessible using `NImageFormatGetBmp` and `NImageFormatGetTiff` functions.

To find out which images formats are supported in the FingerCell EDK in version-independent way `NImageFormatGetFormatCount` and `NImageFormatGetFormat` functions should be used.

Name, file name pattern (file filter) and default file extension of the image format can be retrieved using `NImageFormatGetName`, `NImageFormatGetFileFilter` and `NImageFormatGetDefaultFileExtension` functions.

To find out which image format should be used to read or write a particular file `NImage-`

`FormatSelect` function should be used.

An image can be loaded and saved from/to file or memory buffer using `NImageFormat-LoadImageFromFile`, `NImageFormatLoadImageFromMemory`, `NImageFormat-SaveImageToFile` and `NImageFormatSaveImageToMemory` functions. Note that not all image formats support both reading and writing. Use `NImageFormatCanRead` and/or `NImageFormatCanWrite` function(s) to check if the particular image format does.

3.1.3. Low-Level Image Input-Output

Low-level image I/O in the FingerCell EDK is implemented in Bmp and Tiff modules.

These modules provides functions for loading and saving images in according format (BMP and TIFF).

Those functions can take parameters that precisely control loading and saving of the image in particular formats.

3.2. Fingerprint Scanner Support

Fingerprint scanner support allows live scanning of fingerprint for further features extraction. Scanner support is implemented in a library on Linux (Scanners API for Linux OS).

The scanners supported by the FingerCell EDK are listed in Table 3.1, “Supported scanners for different platforms”.

Table 3.1. Supported scanners for different platforms

Scanner	Linux
AuthenTec AES4000	x
AuthenTec AF-S2	x
Fujitsu MBF200	x
Shimizu/Tacoma CMOS	x
Startek FM200	x

3.3. Fingerprint Scanners API for Linux OS

Below `scanner_info` structure is described:

```
struct scanner_info
{
        int dpi;
        char *name;
        int version;
```



```
        void (*close) (void);
        int (*init) (void);
        unsigned char * (*read) (int *width, int *height);
        int timeout;
        int max_bad_area;
        int lfd;
};
```

Each scanner driver has an exported name pointing to *scanner_info* structure. This structure is the same for every scanner.

scanner_info structure description:

Structure member	Description
int *dpi*	Specifies scanners resolution in DPI.
char **name*	Scanner name in human readable format (null-terminated string).
int *version*	Driver version.
void (**close*) (void)	Function used to close hardware device after use.
int (**init*) (void)	Function used to open hardware device before use. Returns 0 if operation was successful.
unsigned char * (**read*) (int *width, int *height)	Image scanning. Returns *NULL*, if no image could be received from scanner or returns pointer to image if scan was successful. Width and height can't be NULL, returns image size. **Remarks** Returned memory can't be released, and is overwritten with second scan.
int timeout;	User adjustable timeout for USB requests in milliseconds.
int max_bad_area;	The estimation of scanned image is made, according to this value. The interval of this value is from 0 to 100 and it means the amount of non-fingerprint data in the image. The driver compares this value to that calcu-

	lated for the scanned image and decides if the image is fingerprint. Drivers default for this variable is 85%. The value may be changed at any time during driver execution, depending on user needs. Reasonable values are 65%-85%. NOTE: This option only has sense for Shimizu/Tacoma driver.
int lfd;	Live finger detection. 0 - disable, 1 - enable. Default: disable

Important

These drivers are not thread safe.

Table 3.2. Requirements of Fingerprints Scanners drivers

Scanner	**Structure `scanner_info` name**	**Requirements**
AuthenTec AES4000	scanner_aes4000	libusb, libusbw [1]
AuthenTec AF-S2	scanner_af_s2	libusb, libusbw [1]
Shimizu/Tacoma CMOS	scanner_shimizu	libusb, libusbw [1]
Fujitsu MBF200	scanner_mbf200	libusb, libusbw [1]
Startek FM200	scanner_fm200	libusb, libusbw [1]
BiometriKa FX 2000/FX 3000/HiScan	scanner_fx2k	libm, and kernel module [2]
Futronic FS80	scanner_fs80	libusb, libgcc_s, libftrScanAPI [3]
Biolink U-Match BI	scanner_u_match	libusb, libgcc_s, libftrScanAPI [3]
SecuGen Hamster III	scanner_hamster3	libusb, libstdc++, libvenus, libvenusdrv [4]

[1] You need to compile libusbw.o and link it with program witch will use this driver.
[2] You need to extract and build kernel module for fx2k scanner from install/Linux_x86/fxdriver-1.04.tar.gz
[3] You can found the libftrScanAPI.so library in the archive install/Linux_x86/ftrScanAPI.tar.gz.
[3] You can found the libftrScanAPI.so library in the archive install/Linux_x86/ftrScanAPI.tar.gz.

[4] For libvenus and libvenusdrv you need to install venus libraries. For instalation you need extract install/Linux_x86/venus-1.2.2.tar.gz and run as root `./install.sh`.

Example:

```
#include <stdlib.h>
#include <scanner.h>
struct scanner_info* scanner = &scanner_fm200;
int main (void)
{
int w, h;
char *image;

// open device
if (scanner->init())
return 1; // initialization error

// scan image
image = scanner->read(&w, &h);
if (image == NULL)
        return 2; // scanning error

// close device
scanner->close();

// return OK
return 0;
}
```

This sample can be compiled using GCC compiler:

```
gcc main.c -Iscanner/include -Lscanner/lib -o sample -lusb -lfpscan
```


Chapter 4. Samples

Purpose of sample applications is to help with using FingerCell EDK. They can be run to get started with FingerCell. Next FingerCell can be learned and evaluated further by modifying and experimenting with their source code.

4.1. Pocket PC 2003 Sample

Sample applications has six possible working modes (user can change mode during application run):

Mode	Description
Enrollment	Default mode. Fingerprint enrollment mode. Loaded fingerprint image will be enrolled into database.
Enrollment with features generalization	Fingerprint enrollment mode. In this mode user must load three fingerprint images. These images are preceded and extracted features are generalized. Generalization process eliminates noised features and makes fingerprint features collection more reliable. After generalization generalized features collection is stored in database.
Verification	Two loaded fingerprint images will be compared in this mode.
Fast Verification	Same as Verification mode but fast extraction is used.
Identification	Fingerprint identification mode. Program will start identification process if you will load fingerprint image.
Fast Identification	Same as Identification mode but fast extraction is used.

According to the chosen mode demonstration program will call different FingerCell functions.

Sample settings (image resolution, matching, extraction and generalization parameters) can be changed using settings dialog: Tools->Options.

4.2. Linux console sample applications

These are a simple demonstration programs for the FingerCell-2.0 library. They are intended to demonstrate the basic use of the FingerCell interface and to check that FingerCell is working.

The `fc_console_demo` reads images from `SampleImages/` directory, processes them with the help of FingerCell and outputs info about the results to a terminal. Source code shows how to perform extraction, fast estraction, verification and identification, it also shows how to perform template generalization (used to enforce better template quality). The demo reads images from TIF files using NImage library.

The `fc_console_list_demo` enrolls fingerprints specified in `db_list.txt` file into memory, next each fingerprint specified in `ident_list.txt` is compared with DB. The timings for feature extraction, fast feature extraction and matching is shown, so this program can be used to evaluate performance of FingerCell, by specifying custom images. Source code shows how to set some FingerCell thresholds, and get information about templates. It also is example of how to use images without NImage library (for images from scanner or files of simple custom format). The demo reads images from PGM files of 8 bits depth. For information about image conversion to PGM files please see Section 4.2.3, “Converting images from TIFF to PGM”.

4.2.1. Compiling

To compile the program with native compiler run:

```
make
```

If cross compiler is used instead, compiler prefix must be defined:

```
make CROSS=arm-linux-
```

> **Note**
> This assumes cross-toolchain is installed and arm-linux-gcc is runable.

To clean up object files, run:

```
make clean
```

4.2.2. Running

To run the program, copy compiled binary file (`fc_console_demo` or `fc_console_list_demo`), required library files (`lib/Linux_arm/FingerCell/`) and the `SampleImages/` directory to the target system and execute it. `fc_console_list_demo` needs also `db_list.txt` and `ident_list.txt` files.

> **Note**
> Libraries must be in the standard directory known by dynamic linker (usually `/usr/lib/` or `/lib/`) alternatively extra paths to libraries may be given in environment variable `LD_LIBRARY_PATH`:


```
LD_LIBRARY_PATH=${PATH_TO_DIR_WITH_LIBRARY_FILES} ./fc_console_demo
```

4.2.3. Converting images from TIFF to PGM

PGM format is just raw pixel data (in binary or text form) with plain text header (please see PGM format specification for more info[1]). Files can be easily converted with either ImageMagic[2] (on Linux) or IrfanView[3] (on Windows). Demo uses images of 8 bits depth (0-255 gray value pixel per byte), please remember to indicate this to conversion tool.

Conversion procedure on Linux:

```
1. Install ImageMagic
2. "cd ../Sample.Linux"
3. For bash, run:
        for f in *.tif ; do
                convert +compress -depth 8 $f `basename $f .tif`.pgm
        done
   For csh, run:
        foreach f ( *.tif )
                convert +compress -depth 8 $f `basename $f .tif`.pgm
        end
```

Conversion procedure on Windows:

```
1. Install and start IrfanView
2. Use "File>Batch Conversion/Rename..." menu
3. Specify 8 bits depth: mark "Use advanced options" check-box; in
   "Set advanced options" window select "CHANGE COLOR DEPTH:" and
   "256 Colors (8BPP)"
```

[1] http://netpbm.sourceforge.net/doc/pgm.html
[2] http://www.imagemagick.org/script/index.php
[3] http://www.irfanview.com/

Chapter 5. Reference (C/C++)

This chapter contains reference of all libraries included in FingerCell EDK for developers using C/C++ language.

Libraries

NCore	Provides infrastructure for Neurotechnologija components.
NFRecord	Provides functionality for packing, unpacking and editing Neurotechnologija Finger Records.
NImages	Provides functionality for loading, saving and converting images in various formats.
FncExtractor	Provides functionality for extracting Neurotechnologija Finger Records from fingerprint images using FingerCell algorithm.
FncMatcher	Provides functionality for comparing Neurotechnologija Finger Records using FaceCell algorithm.

5.1. NCore Library

Provides infrastructure for Neurotechnologija components.

Windows

Import library: `NCore.dll.lib`.

DLL: `NCore.dll`.

Requirements:

- Microsoft Visual C++ 2005 Libraries runtime.
- Microsoft Layer for Unicode on Windows 98/ME (`unicows.dll`).

Linux

Shared object: `libNCore.so`.

Modules

NCore	Provides infrastructure/basic functionality for Neurotechnologija components.
NErrors	Defines error codes used in Neurotechnologija components.
NMemory	Provides memory management for Neurotechnologija components.
NParameters	Provides functionality for working with parameters for Neurotechnologija components.
NTypes	Defines types and macros used in Neurotechnologija components.

5.1.1. NCore Module

Provides infrastructure/basic functionality for Neurotechnologija components.

Header file: `NCore.h`.

See Also

NCore Library

5.1.2. NErrors Module

Defines error codes used in Neurotechnologija components.

Header file: `NErrors.h`.

Macros

-10	N_E_ARGUMENT	Argument is invalid.
-11	N_E_ARGUMENT_NULL	Argument value is NULL where non-NULL value was expected.
-12	N_E_ARGUMENT_OUT_OF_RANGE	Argument value is out of range.
-2	N_E_CORE	Standard error has occurred (for internal use).
-15	N_E_END_OF_STREAM	Attempted to read file or buffer after its end.
-1	N_E_FAILED	Unspecified error has occurred.

-13	N_E_FORMAT	Format of argument value is invalid.
-9	N_E_INDEX_OUT_OF_RANGE	Index is out of range (for internal use).
-7	N_E_INVALID_OPERATION	Attempted to perform invalid operation.
-14	N_E_IO	Input/output error has occurred.
-5	N_E_NOT_IMPLEMENTED	Functionality is not implemented.
-200	N_E_NOT_REGISTERED	Module is not registered.
-6	N_E_NOT_SUPPORTED	Functionality is not supported.
-3	N_E_NULL_REFERENCE	Null reference has occurred (for internal use).
-4	N_E_OUT_OF_MEMORY	There were not enough memory.
-8	N_E_OVERFLOW	Arithmetic overflow has occurred.
-100	N_E_PARAMETER	Parameter ID is invalid.
-101	N_E_PARAMETER_READ_ONLY	Attempted to set read only parameter.
0	N_OK	No error.
	NFailed	Determines whether function result indicates error.
	NSucceeded	Determines whether function result indicates success.

See Also

NCore Library

5.1.3. NMemory Module

Provides memory management for Neurotechnologija components.

Header file: `NMemory.h`.

Functions

`NAlloc`	Allocates memory block.
`NCAlloc`	Allocates memory block with all bytes set to zero.

`NCompare`	Compares bytes in two memory blocks.
`NCopy`	Copies data between memory blocks.
`NFill`	Sets bytes of memory block to specified value.
`NFree`	Deallocates memory block.
`NMove`	Move data from one memory block to another.
`NReAlloc`	Reallocates memory block.

Macros

NClear	Clears memory block.

See Also

NCore Library

5.1.4. NParameters Module

Provides functionality for working with parameters for Neurotechnologija components.

Header file: `NParameters.h`.

Macros

N_PC_TYPE_ID	Specifies that type id (NInt value, one of N_TYPE_XXX) of the parameter should be retrieved.
NParameterMakeId	Makes parameter id.
N_TYPE_BOOL	Specifies that parameter type is NBool.
N_TYPE_BYTE	Specifies that parameter type is NByte.
N_TYPE_CHAR	Specifies that parameter type is NChar.
N_TYPE_DOUBLE	Specifies that parameter type is NDouble.
N_TYPE_FLOAT	Specifies that parameter type is NFloat.
N_TYPE_INT	Specifies that parameter type is NInt.

N_TYPE_LONG	Specifies that parameter type is NLong.
N_TYPE_SBYTE	Specifies that parameter type is NSByte.
N_TYPE_SHORT	Specifies that parameter type is NShort.
N_TYPE_STRING	Specifies that parameter type is null-terminated string of NChar.
N_TYPE_UINT	Specifies that parameter type is NUInt.
N_TYPE_ULONG	Specifies that parameter type is NULong.
N_TYPE_USHORT	Specifies that parameter type is NUShort.

See Also

NCore Library

5.1.4.1. NParameterMakeId Macro

Makes parameter id.

```
#define NParameterMakeId(code, index, id)
```

Parameters

code	One of N_PC_XXX.
index	Reserved, must be zero.
id	One of the parameter ids provided by a Neurotechnologija module.

See Also

NParameters Module

5.1.5. NTypes Module

Defines types and macros used in Neurotechnologija components.

Header file: NTypes.h.

Structures

NIndexPair	Represents a pair of indexes.
NRational	Represents a signed rational number.
NURational	Represents an unsigned rational number.

Enumerations

NByteOrder	Specifies byte order.
NFileAccess	Specifies access to a file.

Types

NAChar	ANSI character (8-bit).
NBool	Same as NBoolean.
NBoolean	32-bit boolean value. See also NTrue and NFalse.
NByte	Same as NUInt8.
NChar	Character type. Either NAChar or NWChar (if N_UNICODE is defined).
NDouble	Double precision floating point number.
NFloat	Same as NSingle.
NHandle	Pointer to unspecified data (same as void *).
NInt	Same as NInt32.
NInt8	8-bit signed integer (signed byte).
NInt16	16-bit signed integer (short).
NInt32	32-bit signed integer (int).
NInt64	64-bit signed integer (long). Not available on some 32-bit platforms.
NLong	Same as NInt64.
NPosType	Platform dependent position type. Signed 64-bit (or 32-bit on some platforms) integer on 32-bit platform, signed 64-bit integer on 64-bit platform).

NResult	Result of a function (same as NInt). See also NErrors module.
NSByte	Same as NInt8.
NShort	Same as NInt16.
NSingle	Single precision floating point number.
NSizeType	Platform dependent size type. Unsigned 32-bit integer on 32-bit platform, unsigned 64-bit integer on 64-bit platform.
NUInt	Same as NUInt32.
NUInt8	8-bit unsigned integer (byte).
NUInt16	16-bit unsigned integer (unsigned short).
NUInt32	32-bit unsigned integer (unsigned int).
NUInt64	64-bit unsigned integer (unsigned long). Not available on some 32-bit platforms.
NULong	Same as NUInt64.
NUShort	Same as NUInt16.
NWChar	Unicode character (16-bit).

Macros

N_64	Defined if compiling for 64-bit architecture.
N_ANSI_C	Defined if ANSI C language compliance is enabled in compiler.
N_API	Defines functions calling convention (stdcall on Windows).
N_BIG_ENDIAN	Defined if compiling for big-endian processor architecture.
N_BYTE_MAX	Maximum value for NByte.
N_BYTE_MIN	Minimum value for NByte.
N_CALLBACK	Defined callbacks calling convention (stdcall on Windows).
N_CALLBACK_AW	Picks either ANSI or Unicode (if

	N_UNICODE is defined) version of the callback (with either 'A' or 'W' suffix accordingly).
N_CPP	Defined if compiling as C++ code.
N_DECLARE_HANDLE	Declares handle with specified name.
N_DOUBLE_MAX	Maximum value for NDouble.
N_DOUBLE_MIN	Minimum value for NDouble.
N_FUNC_AW	Picks either ANSI or Unicode (if N_UNICODE is defined) version of the function (with either 'A' or 'W' suffix accordingly).
N_GCC	Defined if compiling with GCC.
N_FLOAT_MAX	Maximum value for NFloat.
N_FLOAT_MIN	Minimum value for NFloat.
N_INT_MAX	Maximum value for NInt.
N_INT_MIN	Minimum value for NInt.
N_INT8_MAX	Maximum value for NInt8.
N_INT8_MIN	Minimum value for NInt8.
N_INT16_MAX	Maximum value for NInt16.
N_INT16_MIN	Minimum value for NInt16.
N_INT32_MAX	Maximum value for NInt32.
N_INT32_MIN	Minimum value for NInt32.
N_INT64_MAX	Maximum value for NInt64.
N_INT64_MIN	Minimum value for NInt64.
N_LIB	Defined if compiling static library.
N_LONG_MAX	Maximum value for NLong.
N_LONG_MIN	Minimum value for NLong.
N_MAC	Defined if compiling for Mac OS.
N_MSVC	Defined if compiling with Microsoft Visual C++.

N_NO_ANSI_FUNC	Defined if compiling for platform without ANSI versions of the functions support.
N_NO_FLOAT	Defined if compiling for platform without floating-point support.
N_NO_INT_64	Defined if compiling for platform without 64-bit integer types support.
N_NO_UNICODE	Defined if compiling without Unicode support.
N_POS_TYPE_MIN	Minimum value for NPosType.
N_POS_TYPE_MAX	Maximum value for NPosType.
N_SBYTE_MAX	Maximum value for NSByte.
N_SBYTE_MIN	Minimum value for NSByte.
N_SHORT_MAX	Maximum value for NShort.
N_SHORT_MIN	Minimum value for NShort.
N_SINGLE_MAX	Maximum value for NSingle.
N_SINGLE_MIN	Minimum value for NSingle.
N_SIZE_TYPE_MIN	Minimum value for NSizeType.
N_SIZE_TYPE_MAX	Maximum value for NSizeType.
N_STRUCT_AW	Picks either ANSI or Unicode (if N_UNICODE is defined) version of the struct (with either 'A' or 'W' suffix accordingly).
N_T	Makes either ANSI or Unicode (if N_UNICODE is defined) string or character constant.
N_UINT_MAX	Maximum value for NUInt.
N_UINT_MIN	Minimum value for NUInt.
N_UINT8_MAX	Maximum value for NUInt8.
N_UINT8_MIN	Minimum value for NUInt8.
N_UINT16_MAX	Maximum value for NUInt16.
N_UINT16_MIN	Minimum value for NUInt16.
N_UINT32_MAX	Maximum value for NUInt32.

N_UINT32_MIN	Minimum value for NUInt32.
N_UINT64_MAX	Maximum value for NUInt64.
N_UINT64_MIN	Minimum value for NUInt64.
N_ULONG_MAX	Maximum value for NULong.
N_ULONG_MIN	Minimum value for NULong.
N_UNICODE	Defined if compiling with Unicode character set (affects NChar type).
N_USHORT_MAX	Maximum value for NUShort.
N_USHORT_MIN	Minimum value for NUShort.
N_WINDOWS	Defined if compiling for Windows.
NULL	Null value for pointer.
NFalse	False value for NBoolean.
NIsReverseByteOrder	Checks if specified byte order is reverse to system byte order.
NTrue	True value for NBoolean.

See Also

NCore Library

5.1.5.1. NByteOrder Enumeration

Specifies byte order.

```
typedef enum NByteOrder_ { } NByteOrder;
```

Members

`nboBigEndian`	Big-endian byte order.
`nboLittleEndian`	Little-endian byte order.
`nboSystem`	System-dependent byte order (either little-endian or big-endian).

See Also

NTypes Module

5.1.5.2. NFileAccess Enumeration

Specifies access to a file.

```
typedef enum NFileAccess_ { } NFileAccess;
```

Members

`nfaRead`	Read access to the file.
`nfaReadWrite`	Read and write access to the file.
`nfaWrite`	Write access to the file.

See Also

NTypes Module

5.1.5.3. NIndexPair Structure

Represents a pair of indexes.

```
typedef struct NIndexPair_ { } NIndexPair;
```

Fields

`Index1`	First index of this NIndexPair.
`Index2`	Second index of this NIndexPair.

See Also

NTypes Module

5.1.5.3.1. NIndexPair.Index1 Field

First index of this NIndexPair.

```
NInt Index1;
```

See Also

NIndexPair

5.1.5.3.2. NIndexPair.Index2 Field

Second index of this NIndexPair.

```
NInt Index2;
```

See Also

NIndexPair

5.1.5.4. NRational Structure

Represents a signed rational number.

```
typedef struct NRational_ { } NRational;
```

Fields

Denominator	Denominator of this NRational.
Numerator	Numerator of this NRational.

See Also

NTypes Module

5.1.5.4.1. NRational.Denominator Field

Denominator of this NRational.

```
NInt Denominator;
```

See Also

NRational

5.1.5.4.2. NRational.Numerator Field

Numerator of this NRational.

```
NInt Numerator;
```

See Also

NRational

5.1.5.5. NURational Structure

Represents an unsigned rational number.

```
typedef struct NURational_ { } NURational;
```

Fields

Denominator	Denominator of this NURational.
Numerator	Numerator of this NURational.

See Also

NTypes Module

5.1.5.5.1. NURational.Denominator Field

Denominator of this NURational.

```
NUInt Denominator;
```

See Also

NURational

5.1.5.5.2. NURational.Numerator Field

Numerator of this NURational.

```
NUInt Numerator;
```

See Also

NURational

5.1.6. NGeometry Module

Provides definitions of geometrical structures types.

Header file: `NGeometry.h` (includes `NTypes.h`).

Structures

NPoint	Structure defining point coordinates in 2D space.
NSize	Structure defining rectangle size.
NRect	Structure defining a rectangle figure in 2D space.

See Also

NCore Library

5.1.6.1. NPoint structure

Structure defining point coordinates in 2D space.

```
typedef struct NPoint_ { } NPoint;
```

Fields

X	Point coordinate on x axis.
Y	Point coordinate on y axis.

See Also

NGeometry

5.1.6.1.1. NPoint.X Field

Point coordinate on x axis.

```
NInt X;
```

See Also

NPoint

5.1.6.1.2. NPoint.Y Field

Point coordinate on y axis.

```
NInt Y;
```


See Also

NPoint

5.1.6.2. NSize structure

Structure defining rectangle size.

```
typedef struct NSize_ { } NSize;
```

Fields

Width	Width.
Height	Height.

See Also

NGeometry

5.1.6.2.1. NSize.Width Field

Width.

```
NInt Width;
```

See Also

NSize

5.1.6.2.2. NSize.Height Field

Height.

```
NInt Height;
```

See Also

NSize

5.1.6.3. NRect structure

Structure defining a rectangle figure in 2D space.

```
typedef struct NRect_ { } NRect;
```


Fields

X	Upper left rectangle corner coordinate on x axis.
Y	Upper left rectangle corner coordinate on y axis.
Width	Rectangle width.
Height	Rectangle height.

See Also

NGeometry

5.1.6.3.1. NRect.X Field

Upper left rectangle corner coordinate on x axis.

```
NInt X;
```

See Also

NRect

5.1.6.3.2. NRect.Y Field

Upper left rectangle corner coordinate on y axis.

```
NInt Y;
```

See Also

NRect

5.1.6.3.3. NRect.Width Field

Rectangle width.

```
NInt Width;
```

See Also

NRect

5.1.6.3.4. NRect.Height Field

Rectangle height.

```
NInt Height;
```

See Also

NRect

5.2. NFRecord Library

Provides functionality for packing, unpacking and editing Neurotechnologija Finger Records.

Windows

Import library: `NFRecord.dll.lib`.

DLL: `NFRecord.dll`.

Requirements:

- `NCore.dll`.

Linux

Shared object: `libNFRecord.so`.

Requirements:

- `libNCore.so`.

Modules

NFRecord	Provides functionality for packing, unpacking and editing Neurotechnologija Finger Records (NFRecords).

5.2.1. NFRecord Module

Provides functionality for packing, unpacking and editing Neurotechnologija Finger Records (NFRecords).

Header file: `NFRecord.h` and `NFRecordV1.h`

Functions

NFRecordAddCore	Adds a NFCore to the end of NFRecord cores.
NFRecordAddDelta	Adds a NFDelta to the end of NFRecord deltas.
NFRecordAddDoubleCore	Adds a NFDoubleCore to the end of NFRecord double cores.
NFRecordAddMinutia	Adds a NFMinutia to the end of NFRecord minutiae.
NFRecordCheck	Checks if format of the packed NFRecord is correct.
NFRecordClearCores	Removes all cores from the NFRecord.
NFRecordClearDeltas	Removes all deltas from the NFRecord.
NFRecordClearDoubleCores	Removes all double cores from the NFRecord.
NFRecordClearMinutiae	Removes all minutiae from the NFRecord.
NFRecordClone	Creates a copy of the NFRecord.
NFRecordCreate	Creates an empty NFRecord.
NFRecordCreateFromMemory	Unpacks a NFRecord from the specified memory buffer.
NFRecordFree	Deletes the NFRecord. After the object is deleted the specified handle is no longer valid.
NFRecordGetCbeffProductType	Retrieves the Cbeff product type of the NFRecord.
NFRecordGetCbeffProductTypeMem	Retrieves the Cbeff product type of the packed NFRecord.
NFRecordGetCore	Retrieves the core at the specified index of the NFRecord.
NFRecordGetCoreCapacity	Retrieves the number of cores that the NFRecord can contain.
NFRecordGetCoreCount	Retrieves the number of cores in the NFRecord.
NFRecordGetCores	Copies all cores of NFRecord to the spe-

	cified array.
NFRecordGetDelta	Retrieves the delta at the specified index of the NFRecord.
NFRecordGetDeltaCapacity	Retrieves the number of deltas that the NFRecord can contain.
NFRecordGetDeltaCount	Retrieves the number of deltas in the NFRecord.
NFRecordGetDeltas	Copies all deltas of NFRecord to the specified array.
NFRecordGetDoubleCore	Retrieves the double core at the specified index of the NFRecord.
NFRecordGetDoubleCoreCapacity	Retrieves the number of double cores that the NFRecord can contain.
NFRecordGetDoubleCoreCount	Retrieves the number of double cores in the NFRecord.
NFRecordGetDoubleCores	Copies all double cores of NFRecord to the specified array.
NFRecordGetG	Retrieves the G of the NFRecord.
NFRecordGetGMem	Retrieves the G of the packed NFRecord.
NFRecordGetHeight	Retrieves the height of the image the NFRecord is made from.
NFRecordGetHeightMem	Retrieves the height of the image the packed NFRecord is made from.
NFRecordGetHorzResolution	Retrieves the horizontal resolution of the image the NFRecord is made from.
NFRecordGetHorzResolutionMem	Retrieves the horizontal resolution of the image the packed NFRecord is made from.
NFRecordGetImpressionType	Retrieves the impression type of the NFRecord.
NFRecordGetImpressionTypeMem	Retrieves the impression type of the packed NFRecord.
NFRecordGetMaxSize	Retrieves the maximal size of a packed NFRecord containing the specified number of minutiae, cores, deltas and double cores and the specified ridge counts type.

NFRecordGetMaxSizeV1	Retrieves the maximal size of a packed in version 1.0 format NFRecord containing the specified number of minutiae, cores, deltas and double cores and the specified ridge counts type.
NFRecordGetMinutia	Retrieves the minutia at the specified index of the NFRecord.
NFRecordGetMinutiaCapacity	Retrieves the number of minutiae that the NFRecord can contain.
NFRecordGetMinutiaCount	Retrieves the number of minutiae in the NFRecord.
NFRecordGetMinutiaFormat	Retrieves the format of the minutiae in NFRecord.
NFRecordGetMinutiaNeighbor	Retrieves the minutia neighbor at the specified index of the minutia at the specified index of the NFRecord.
NFRecordGetMinutiaNeighborCount	Retrieves the number of minutia neighbors in the minutia at the specified index of the NFRecord.
NFRecordGetMinutiaNeighbors	Copies all minutia neighbors of the minutia at the specified index of the NFRecord to the specified array.
NFRecordGetMinutiae	Copies all minutiae of NFRecord to the specified array.
NFRecordGetPatternClass	Retrieves the pattern class of the NFRecord.
NFRecordGetPatternClassMem	Retrieves the pattern class of the packed NFRecord.
NFRecordGetPosition	Retrieves the finger position of the NFRecord.
NFRecordGetPositionMem	Retrieves the finger position of the packed NFRecord.
NFRecordGetQuality	Retrieves the quality of the NFRecord.
NFRecordGetQualityMem	Retrieves the quality of the packed NFRecord.
NFRecordGetRidgeCountsType	Retrieves the ridge counts type the NFRecord contains.

NFRecordGetSize	Calculates packed size of the NFRecord.
NFRecordGetSizeV1	Calculates packed in version 1.0 format size of the NFRecord.
NFRecordGetVertResolution	Retrieves the vertical resolution of the image the NFRecord is made from.
NFRecordGetVertResolutionMem	Retrieves the vertical resolution of the image the packed NFRecord is made from.
NFRecordGetWidth	Retrieves the width of the image the NFRecord is made from.
NFRecordGetWidthMem	Retrieves the width of the image the packed NFRecord is made from.
NFRecordInfoDispose	For internal use.
NFRecordInsertCore	Inserts a NFCore into the NFRecord at the specified index.
NFRecordInsertDelta	Inserts a NFDelta into the NFRecord at the specified index.
NFRecordInsertDoubleCore	Inserts a NFDoubleCore into the NFRecord at the specified index.
NFRecordInsertMinutia	Inserts a NFMinutia into the NFRecord at the specified index.
NFRecordRemoveCore	Removes the core at the specified index of the NFRecord.
NFRecordRemoveDelta	Removes the delta at the specified index of the NFRecord.
NFRecordRemoveDoubleCore	Removes the double core at the specified index of the NFRecord.
NFRecordRemoveMinutia	Removes the minutia at the specified index of the NFRecord.
NFRecordSaveToMemory	Packs the NFRecord into the specified memory buffer.
NFRecordSaveToMemoryV1	Packs the NFRecord into the specified memory buffer in version 1.0 format.
NFRecordSetCbeffProductType	Sets the Cbeff product type.
NFRecordSetCore	Sets a NFCore at the specified index of the NFRecord.

NFRecordSetCoreCapacity	Sets the number of cores that the NFRecord can contain.
NFRecordSetDelta	Sets a NFDelta at the specified index of the NFRecord.
NFRecordSetDeltaCapacity	Sets the number of deltas that the NFRecord can contain.
NFRecordSetDoubleCore	Sets a NFDoubleCore at the specified index of the NFRecord.
NFRecordSetDoubleCoreCapacity	Sets the number of double cores that the NFRecord can contain.
NFRecordSetG	Sets the G of the NFRecord.
NFRecordSetImpressionType	Sets the impression type of the NFRecord.
NFRecordSetMinutia	Sets a NFMinutia at the specified index of the NFRecord.
NFRecordSetMinutiaCapacity	Sets the number of minutiae that the NFRecord can contain.
NFRecordSetMinutiaNeighbor	Sets a NFMinutiaNeighbor at the specified index of the minutia at the specified index of the NFRecord.
NFRecordSetMinutiaFormat	Sets the format of the minutiae in NFRecord.
NFRecordSetPatternClass	Sets the pattern class of the NFRecord.
NFRecordSetPosition	Sets the finger position of the NFRecord.
NFRecordSetQuality	Sets the quality of the NFRecord.
NFRecordSetRidgeCountsType	Sets the ridge counts type the NFRecord contains.
NFRecordSortMinutiae	Sorts minutiae in NFRecord by the specified order.
NFRecordTruncateMinutiae	Truncates minutiae in NFRecord by peeling off the minutiae convex hull while minutia count is greater than the specified maximal count.
NFRecordTruncateMinutiaeByQuality	Truncates minutiae in NFRecord by removing minutiae which NFMinutia.Quality field value is less than the specified threshold

	while minutia count is greater than the specified maximal count.

Structures

NFCore	Represents a core in a NFRecord.
NFDelta	Represents a delta in a NFRecord.
NFDoubleCore	Represents a double core in a NFRecord.
NFMinutia	Represents a minutia in a NFRecord.
NFMinutiaNeighbor	Represents a minutia neighbor in a NFRecord.
NFRecordInfo	For internal use.

Enumerations

NFImpressionType	Specifies the impression type.
NFMinutiaFormat	Specifies the minutia format.
NFMinutiaOrder	Specifies minutia order.
NFMinutiaType	Specifies the minutia type.
NFPatternClass	Specifies the pattern class of the fingerprint.
NFPosition	Specifies the finger position.
NFRidgeCountsType	Specifies the type of ridge counts contained in a NFRecord.

Types

HNFRecord	Handle to NFRecord object.

Macros

NFR_BLOCK_SIZE	For internal use.
NFR_MAX_BLOCKED_ORIENTS_DIME	For internal use.

NSION	
NFR_MAX_CORE_COUNT	The maximum number of cores a NFRecord can contain.
NFR_MAX_DELTA_COUNT	The maximum number of deltas a NFRecord can contain.
NFR_MAX_DIMENSION	The maximum value for x and y coordinates of a minutia, core, delta or double core in a NFRecord.
NFR_MAX_DOUBLE_CORE_COUNT	The maximum number of double cores a NFRecord can contain.
NFR_MAX_MINUTIA_COUNT	The maximum number of minutiae a NFRecord can contain.
NFR_RESOLUTION	The resolution of minutiae, cores, deltas and double cores coordinates in a NFRecord.
NFR_SAVE_BLOCKED_ORIENTS	The flag indicating whether blocked orientations should be packed in NFRecord.
NFR_SKIP_BLOCKED_ORIENTS	The flag indicating whether blocked orientations should be skipped while unpacking NFRecord.
NFR_SKIP_CURVATURES	The flag indicating whether minutiae curvatures should be skipped while unpacking or packing NFRecord.
NFR_SKIP_GS	The flag indicating whether minutiae gs should be skipped while unpacking or packing NFRecord.
NFR_SKIP_QUALITIES	The flag indicating whether minutiae qualities should be skipped while unpacking or packing NFRecord.
NFR_SKIP_RIDGE_COUNTS	The flag indicating whether ridge counts should be skipped while unpacking or packing NFRecord.
NFR_SKIP_SINGULAR_POINTS	The flag indicating whether singular points (cores, deltas and double cores) should be skipped while unpacking or packing NFRecord.

See Also

NFRecord Library

5.2.1.1. NFCore Structure

Represents a core in a NFRecord.

```
typedef struct NFCore_ { } NFCore;
```

Fields

Angle	Angle of this NFCore.
X	X coordinate of this NFCore.
Y	Y coordinate of this NFCore.

See Also

NFRecord Module

5.2.1.1.1. NFCore.Angle Field

Angle of this NFCore.

```
NInt Angle;
```

Remarks

The angle of the core is specified in 180/128 degrees units in counterclockwise order and can not be less than zero or greater than 256 minus one.

The value of -1 can be specified if the angle of the core is unknown.

See Also

NFCore

5.2.1.1.2. NFCore.X Field

X coordinate of this NFCore.

```
NUShort X;
```

Remarks

The x coordinate of the core is specified in pixels at NFR_RESOLUTION and *X* *

[NFRecord horizontal resolution] / NFR_RESOLUTION can not be greater than NFR_MAX_DIMENSION or NFRecord width minus one.

See Also

NFCore

5.2.1.1.3. NFCore.Y Field

Y coordinate of this NFCore.

```
NUShort Y;
```

Remarks

The y coordinate of the core is specified in pixels at NFR_RESOLUTION and *Y* * [NFRecord vertical resolution] / NFR_RESOLUTION can not be greater than NFR_MAX_DIMENSION or NFRecord height minus one.

See Also

NFCore

5.2.1.2. NFDelta Structure

Represents a delta in a NFRecord.

```
typedef struct NFDelta_ { } NFDelta;
```

Fields

Angle1	First angle of this NFDelta.
Angle2	Second angle of this NFDelta.
Angle3	Third angle of this NFDelta.
X	X coordinate of this NFDelta.
Y	Y coordinate of this NFDelta.

See Also

NFRecord Module

5.2.1.2.1. NFDelta.Angle1 Field

First angle of this NFDelta.


```
NInt Angle1;
```

Remarks

The angle of the delta is specified in 180/128 degrees units in counterclockwise order and can not be less than zero or greater than 256 minus one.

The value of -1 can be specified if the first angle of the delta is unknown.

See Also

NFDelta

5.2.1.2.2. NFDelta.Angle2 Field

Second angle of this NFDelta.

```
NInt Angle2;
```

Remarks

The angle of the delta is specified in 180/128 degrees units in counterclockwise order and can not be less than zero or greater than 256 minus one.

The value of -1 can be specified if the second angle of the delta is unknown.

See Also

NFDelta

5.2.1.2.3. NFDelta.Angle3 Field

Third angle of this NFDelta.

```
NInt Angle3;
```

Remarks

The angle of the delta is specified in 180/128 degrees units in counterclockwise order and can not be less than zero or greater than 256 minus one.

The value of -1 can be specified if the third angle of the delta is unknown.

See Also

NFDelta

5.2.1.2.4. NFDelta.X Field

X coordinate of this NFDelta.

```
NUShort X;
```

Remarks

The x coordinate of the delta is specified in pixels at NFR_RESOLUTION and *X* * [NFRecord horizontal resolution] / NFR_RESOLUTION can not be greater than NFR_MAX_DIMENSION or NFRecord width minus one.

See Also

NFDelta

5.2.1.2.5. NFDelta.Y Field

Y coordinate of this NFDelta.

```
NUShort Y;
```

Remarks

The y coordinate of the delta is specified in pixels at NFR_RESOLUTION and *Y* * [NFRecord vertical resolution] / NFR_RESOLUTION can not be greater than NFR_MAX_DIMENSION or NFRecord height minus one.

See Also

NFDelta

5.2.1.3. NFDoubleCore Structure

Represents a double core in a NFRecord.

```
typedef struct NFDoubleCore_ { } NFDoubleCore;
```

Fields

X	X coordinate of this NFDoubleCore.
Y	Y coordinate of this NFDoubleCore.

See Also

NFRecord Module

5.2.1.3.1. NFDoubleCore.X Field

X coordinate of this NFDoubleCore.

```
NUShort X;
```

Remarks

The x coordinate of the double core is specified in pixels at NFR_RESOLUTION and *X* * [NFRecord horizontal resolution] / NFR_RESOLUTION can not be greater than NFR_MAX_DIMENSION or NFRecord width minus one.

See Also

NFDoubleCore

5.2.1.3.2. NFDoubleCore.Y Field

Y coordinate of this NFDoubleCore.

```
NUShort Y;
```

Remarks

The y coordinate of the double core is specified in pixels at NFR_RESOLUTION and *Y* * [NFRecord vertical resolution] / NFR_RESOLUTION can not be greater than NFR_MAX_DIMENSION or NFRecord height minus one.

See Also

NFDoubleCore

5.2.1.4. NFImpressionType Enumeration

Specifies the impression type.

```
typedef enum NFImpressionType_ { } NFImpressionType;
```

Members

`nfitLatentImpression`	Latent impression fingerprint.
`nfitLatentLift`	Latent lift fingerprint.
`nfitLatentPhoto`	Latent photo fingerprint.
`nfitLatentTracing`	Latent tracing fingerprint.

`nfitLiveScanContactless`	Live-scanned fingerprint using contactless device.
`nfitLiveScanPlain`	Live-scanned plain fingerprint.
`nfitLiveScanRolled`	Live-scanned rolled fingerprint.
`nfitNonliveScanPlain`	Nonlive-scanned (from paper) plain fingerprint.
`nfitNonliveScanRolled`	Nonlive-scanned (from paper) rolled fingerprint.
`nfitSwipe`	Live-scanned fingerprint by sliding the finger across a "swipe" sensor.

Remarks

This enumeration is implemented according to ANSI/NIST-ITL 1-2000 and ANSI INCITS 378-2004 standards.

See Also

NFRecord Module

5.2.1.5. NFMinutia Structure

Represents a minutia in a NFRecord.

```
typedef struct NFMinutia_ { } NFMinutia;
```

Fields

`Angle`	Angle of this NFMinutia.
`Curvature`	Ridge curvature near this NFMinutia.
`G`	G (ridge density) near this NFMinutia.
`Quality`	Quality of this NFMinutia.
`Type`	Type of this NFMinutia.
`X`	X coordinate of this NFMinutia.
`Y`	Y coordinate of this NFMinutia.

See Also

NFRecord Module

5.2.1.5.1. NFMinutia.Angle Field

Angle of this NFMinutia.

```
NByte Angle;
```

Remarks

The angle of the minutia is specified in 180/128 degrees units in counterclockwise order and can not be greater than 256 minus one.

See Also

NFMinutia

5.2.1.5.2. NFMinutia.Curvature Field

Ridge curvature near this NFMinutia.

```
NByte Curvature;
```

Remarks

If curvature of the minutia is unknown it must be set to 255.

See Also

NFMinutia

5.2.1.5.3. NFMinutia.G Field

G (ridge density) near this NFMinutia.

```
NByte G;
```

Remarks

If G of the minutia is unknown it must be set to 255.

See Also

NFMinutia

5.2.1.5.4. NFMinutia.Quality Field

Quality of this NFMinutia.

```
NByte Quality;
```

Remarks

The quality of the minutia must be in the range [0, 100]. The higher it is, the better the quality of the minutia is.

If quality of the minutia is unknown it must be set to zero.

See Also

NFMinutia

5.2.1.5.5. NFMinutia.Type Field

Type of this NFMinutia.

```
NFMinutiaType Type;
```

See Also

NFMinutia

5.2.1.5.6. NFMinutia.X Field

X coordinate of this NFMinutia.

```
NUShort X;
```

Remarks

The x coordinate of the minutia is specified in pixels at NFR_RESOLUTION and *X* * [NFRecord horizontal resolution] / NFR_RESOLUTION can not be greater than NFR_MAX_DIMENSION or NFRecord width minus one.

See Also

NFMinutia

5.2.1.5.7. NFMinutia.Y Field

Y coordinate of this NFMinutia.

```
NUShort Y;
```


Remarks

The y coordinate of the minutia is specified in pixels at NFR_RESOLUTION and *Y* * [NFRecord vertical resolution] / NFR_RESOLUTION can not be greater than NFR_MAX_DIMENSION or NFRecord height minus one.

See Also

NFMinutia

5.2.1.6. NFMinutiaFormat Enumeration

Specifies the minutia format.

This enumeration allows a bitwise combination of its member values.

```
typedef enum NFMinutiaFormat_ { } NFMinutiaFormat;
```

Members

`nfmfHasCurvature`	Indicates that NFMinutia. Curvature field contains meaningful value and is preserved during unpacking/packing of NFRecord.
`nfmfHasG`	Indicates that NFMinutia. G field contains meaningful value and is preserved during unpacking/packing of NFRecord.
`nfmfHasQuality`	Indicates that NFMinutia. Quality field contains meaningful value and is preserved during unpacking/packing of NFRecord.

See Also

NFRecord Module | NFMinutia

5.2.1.7. NFMinutiaNeighbor Structure

Represents a minutia neighbor in a NFRecord.

```
typedef struct NFMinutiaNeighbor_ { } NFMinutiaNeighbor;
```

Fields

`Index`	Index of neighbor minutia.
`RidgeCount`	Ridge count to neighbor minutia.

See Also

NFRecord Module

5.2.1.7.1. NFMinutiaNeighbor.Index Field

Index of neighbor minutia.

```
NInt Index;
```

See Also

NFMinutiaNeighbor

5.2.1.7.2. NFMinutiaNeighbor.RidgeCount Field

Ridge count to neighbor minutia.

```
NByte RidgeCount;
```

See Also

NFMinutiaNeighbor

5.2.1.8. NFMinutiaOrder Enumeration

Specifies minutia order.

```
typedef enum NFMinutiaOrder_ { } NFMinutiaOrder;
```

Members

`nfmoAscending`	Specifies than minutiae are sorted ascending by the specified order.
`nfmoDescending`	Specifies than minutiae are sorted descending by the specified order.
`nfmoCartesianXY`	Specifies than minutiae are sorted by NFMinutia.X field. If NFMinutia.X field of two minutiae are equal NFMinutia.Y field is compared.
`nfmoCartesianYX`	Specifies than minutiae are sorted by NFMinutia.Y field. If NFMinutia.Y field of two minutiae are equal NFMinutia.X is compared.

nfmoAngle	Specifies than minutiae are sorted by NFMinutia.Angle field.
nfmoPolar	Specifies than minutiae are sorted by distance from minutiae center of mass. If distance of two minutiae are equal NFMinutia.Angle field is compared.

See Also

NFRecord Module

5.2.1.9. NFMinutiaType Enumeration

Specifies the minutia type.

```
typedef enum NFMinutiaType_ { } NFMinutiaType;
```

Members

nfmtBifurcation	The minutia that is a bifurcation of a ridge.
nfmtEnd	The minutia that is an end of a ridge.
nfmtUnknown	The type of the minutia is unknown.

See Also

NFRecord Module | NFMinutia

5.2.1.10. NFPatternClass Enumeration

Specifies the pattern class of the fingerprint.

```
typedef enum NFPatternClass_ { } NFPatternClass;
```

Members

nfpcAccidentalWhorl	Accidental whorl pattern class.
nfpcAmputation	Amputation. Pattern class is not available.
nfpcCentralPocketLoop	Central pocket loop pattern class.
nfpcDoubleLoop	Double loop pattern class.

nfpcLeftSlantLoop	Left slant loop pattern class.
nfpcPlainArch	Plain arch pattern class.
nfpcPlainWhorl	Plain whorl pattern class.
nfpcRadialLoop	Radial loop pattern class.
nfpcRightSlantLoop	Right slant loop pattern class.
nfpcScar	Scar. Pattern class is not available.
nfpcTentedArch	Tented arch pattern class.
nfpcUlnarLoop	Ulnar loop pattern class.
nfpcUnknown	Unknown pattern class.
nfpcWhorl	Whorl pattern class.

Remarks

This enumeration is implemented according to ANSI/NIST-ITL 1-2000 standard.

See Also

NFRecord Module

5.2.1.11. NFPosition Enumeration

Specifies the finger position.

```
typedef enum NFPosition_ { } NFPosition;
```

Members

nfpLeftIndex	Index finger of the left hand.
nfpLeftLittle	Little finger of the left hand.
nfpLeftMiddle	Middle finger of the left hand.
nfpLeftRing	Ring finger of the left hand.
nfpLeftThumb	Thumb of the left hand.
nfpRightIndex	Index finger of the right hand.
nfpRightLittle	Little finger of the right hand.

nfpRightMiddle	Middle finger of the right hand.
nfpRightRing	Ring finger of the right hand.
nfpRightThumb	Thumb of the right hand.
nfpUnknown	Unknown finger.

Remarks

This enumeration is implemented according to ANSI/NIST-ITL 1-2000 and ANSI INCITS 378-2004 standards.

See Also

NFRecord Module

5.2.1.12. NFRecordAddCore Function

Adds a NFCore to the end of NFRecord cores.

```
NResult N_API NFRecordAddCore(
        HNFRecord hRecord,
        const NFCore * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[in] Pointer to the NFCore to add.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_INVALID_OPERATION	Number of cores in NFRecord (see NFRecordGetCoreCount) is equal to NFR_MAX_CORE_COUNT.

See Also

NFRecord Module | HNFRecord | NFCore

5.2.1.13. NFRecordAddDelta Function

Adds a NFDelta to the end of NFRecord deltas.

```
NResult N_API NFRecordAddDelta(
        HNFRecord hRecord,
        const NFDelta * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[in] Pointer to the NFDelta to add.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_INVALID_OPERATION	Number of deltas in NFRecord (see NFRecordGetDeltaCount) is equal to NFR_MAX_DELTA_COUNT.

See Also

NFRecord Module | HNFRecord | NFDelta

5.2.1.14. NFRecordAddDoubleCore Function

Adds a NFDoubleCore to the end of NFRecord double cores.

```
NResult N_API NFRecordAddDoubleCore(
        HNFRecord hRecord,
        const NFDoubleCore * pValue
);
```


Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[in] Pointer to the NFDoubleCore to add.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_INVALID_OPERATION	Number of double cores in NFRecord (see NFRecordGetDoubleCoreCount) is equal to NFR_MAX_DOUBLE_CORE_COUNT.

See Also

NFRecord Module | HNFRecord | NFDoubleCore

5.2.1.15. NFRecordAddMinutia Function

Adds a NFMinutia to the end of NFRecord minutiae.

```
NResult N_API NFRecordAddMinutia(
        HNFRecord hRecord,
        const NFMinutia * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[in] Pointer to the NFMinutia to add.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_INVALID_OPERATION	Number of minutiae in NFRecord (see NFRecordGetMinutiaCount) is equal to NFR_MAX_MINUTIA_COUNT.

See Also

NFRecord Module | HNFRecord | NFMinutia

5.2.1.16. NFRecordCheck Function

Checks if format of the packed NFRecord is correct.

```
NResult N_API NFRecordCheck(
        const void * buffer,
        NSizeType bufferSize
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

See Also

NFRecord Module

5.2.1.17. NFRecordClearCores Function

Removes all cores from the NFRecord.

```
NResult N_API NFRecordClearCores(
        HNFRecord hRecord
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord

5.2.1.18. NFRecordClearDeltas Function

Removes all deltas from the NFRecord.

```
NResult N_API NFRecordClearDeltas(
        HNFRecord hRecord
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord

5.2.1.19. NFRecordClearDoubleCores Function

Removes all double cores from the NFRecord.

```
NResult N_API NFRecordClearDoubleCores(
        HNFRecord hRecord
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord

5.2.1.20. NFRecordClearMinutiae Function

Removes all minutiae from the NFRecord.


```
NResult N_API NFRecordClearMinutiae(
        HNFRecord hRecord
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord

5.2.1.21. NFRecordClone Function

Creates a copy of the NFRecord.

```
NResult N_API NFRecordClone(
        HNFRecord hRecord,
        HNFRecord * pHClonedRecord
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pHClonedRecord	[out] Pointer to a HNFRecord that receives handle to newly created NFRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pHClonedRecord* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using NFRecordFree function.

See Also

NFRecord Module | HNFRecord | NFRecordFree

5.2.1.22. NFRecordCreate Function

Creates an empty NFRecord.

```
NResult N_API NFRecordCreate(
        NUShort width,
        NUShort height,
        NUShort horzResolution,
        NUShort vertResolution,
        NUInt flags,
        HNFRecord * pHRecord
);
```

Parameters

width	[in] Specifies width of fingerprint image.
height	[in] Specifies height of fingerprint image.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of fingerprint image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of fingerprint image.
flags	[in] Bitwise combination of zero or more flags that controls behavior of the function. **This parameter is reserved, must be zero**.
pHRecord	[out] Pointer to HNFRecord that receives handle to created NFRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is zero.
N_E_ARGUMENT_NULL	*pHRecord* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using NFRecordFree function.

See Also

NFRecord Module | HNFRecord | NFRecordFree

5.2.1.23. NFRecordCreateFromMemory Function

Unpacks a NFRecord from the specified memory buffer.

```
NResult N_API NFRecordCreateFromMemory(
        const void * buffer,
        NSizeType bufferSize,
        NUInt flags,
        NFRecordInfo * pInfo,
        HNFRecord * pHRecord
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.

flags	[in] Bitwise combination of zero or more flags that controls behavior of the function.
pInfo	[out] For internal use. Must be NULL.
pHRecord	[out] Pointer to HNFRecord that receives handle to newly created NFRecord object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pHRecord* or *buffer* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

The following flags are supported:

- NFR_SKIP_BLOCKED_ORIENTS
- NFR_SKIP_CURVATURES
- NFR_SKIP_GS
- NFR_SKIP_QUALITIES
- NFR_SKIP_RIDGE_COUNTS
- NFR_SKIP_SINGULAR_POINTS

This function supports both NFRecord version 1.0 and 2.0 formats.

Created object must be deleted using `NFRecordFree` function.

See Also

NFRecord Module | HNFRecord | NFRecordInfo | `NFRecordFree` | `NFRecord-SaveToMemory`

5.2.1.24. NFRecordFree Function

Deletes the NFRecord. After the object is deleted the specified handle is no longer valid.

```
void N_API NFRecordFree(
        HNFRecord hRecord
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

Remarks

If *hRecord* is NULL, does nothing.

See Also

NFRecord Module | HNFRecord

5.2.1.25. NFRecordGetCbeffProductType Function

Retrieves the Cbeff product type of the NFRecord.

```
NResult N_API NFRecordGetCbeffProductType(
    HNFRecord hRecord,
    NUShort * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NUShort that receives Cbeff product type.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord | NFRecordSetCbeffProductType NFRecordGetCbeffProductTypeMem

5.2.1.26. NFRecordGetCbeffProductTypeMem Function

Retrieves the Cbeff product type of the packed NFRecord.

```
NResult N_API NFRecordGetCbeffProductTypeMem(
    const void * buffer,
    NSizeType bufferSize,
    NUShort * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.
pValue	[out] Pointer to NUShort that receives Cbeff product type.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord | NFRecordSetCbeffProductType NFRecordGetCbeffProductType

5.2.1.27. NFRecordGetCore Function

Retrieves the core at the specified index of the NFRecord.

```
NResult N_API NFRecordGetCore(
```



```
        HNFRecord hRecord,
        NInt index,
        NFCore * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of core to retrieve.
pValue	[out] Pointer to NFCore that receives core.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to core count obtained using NFRecordGetCoreCount function.

See Also

NFRecord Module | HNFRecord | NFCore | NFRecordGetCoreCount | NFRecordSetCore

5.2.1.28. NFRecordGetCoreCapacity Function

Retrieves the number of cores that the NFRecord can contain.

```
NResult N_API NFRecordGetCoreCapacity(
        HNFRecord hRecord,
        NInt * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NInt that receives number of cores NFRecord can contain.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

Core capacity is the number of cores that the NFRecord can store. Core count (see `NFRecordGetCoreCount` function) is the number of cores that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding cores the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | `NFRecordSetCoreCapacity` | `NFRecordGetCoreCount`

5.2.1.29. NFRecordGetCoreCount Function

Retrieves the number of cores in the NFRecord.

```
NResult N_API NFRecordGetCoreCount(
        HNFRecord hRecord,
        NInt * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NInt that receives number of cores.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

Core capacity (see NFRecordGetCoreCapacity and NFRecordSetCoreCapacity functions) is the number of cores that the NFRecord can store. Core count is the number of cores that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding cores the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | NFRecordGetCoreCapacity | NFRecordSet-CoreCapacity

5.2.1.30. NFRecordGetCores Function

Copies all cores of NFRecord to the specified array.

```
NResult N_API NFRecordGetCores(
        HNFRecord hRecord,
        NFCore * arCores
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
arCores	[out] Pointer to array of NFCore that receives cores.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *arCores* is NULL.

Remarks

Array *arCores* points to must be large enough to receive all NFRecord cores. See NFRecordGetCoreCount function.

See Also

NFRecord Module | HNFRecord | NFCore | NFRecordGetCoreCount

5.2.1.31. NFRecordGetDelta Function

Retrieves the delta at the specified index of the NFRecord.

```
NResult N_API NFRecordGetDelta(
        HNFRecord hRecord,
        NInt index,
        NFDelta * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of delta to retrieve.
pValue	[out] Pointer to NFDelta that receives delta.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to delta count obtained using NFRecordGetDeltaCount function.

See Also

NFRecord Module | HNFRecord | NFDelta | NFRecordGetDeltaCount | NFRecordSetDelta

5.2.1.32. NFRecordGetDeltaCapacity Function

Retrieves the number of deltas that the NFRecord can contain.

```
NResult N_API NFRecordGetDeltaCapacity(
        HNFRecord hRecord,
        NInt * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NInt that receives number of deltas NFRecord can contain.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

Delta capacity is the number of deltas that the NFRecord can store. Delta count (see NFRecordGetDeltaCount function) is the number of deltas that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding deltas the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | NFRecordSetDeltaCapacity | NFRecordGetDeltaCount

5.2.1.33. NFRecordGetDeltaCount Function

Retrieves the number of deltas in the NFRecord.

```
NResult N_API NFRecordGetDeltaCount(
        HNFRecord hRecord,
```



```
        NInt * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NInt that receives number of deltas.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

Delta capacity (see `NFRecordGetDeltaCapacity` and `NFRecordSetDeltaCapacity` functions) is the number of deltas that the NFRecord can store. Delta count is the number of deltas that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding deltas the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | `NFRecordGetDeltaCapacity` | `NFRecordSetDeltaCapacity`

5.2.1.34. NFRecordGetDeltas Function

Copies all deltas of NFRecord to the specified array.

```
NResult N_API NFRecordGetDeltas(
        HNFRecord hRecord,
        NFDelta * arDeltas
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
arDeltas	[out] Pointer to array of NFDelta that receives deltas.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *arDeltas* is NULL.

Remarks

Array *arDeltas* points to must be large enough to receive all NFRecord deltas. See NFRecordGetDeltaCount function.

See Also

NFRecord Module | HNFRecord | NFDelta | NFRecordGetDeltaCount

5.2.1.35. NFRecordGetDoubleCore Function

Retrieves the double core at the specified index of the NFRecord.

```
NResult N_API NFRecordGetDoubleCore(
        HNFRecord hRecord,
        NInt index,
        NFDoubleCore * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of double core to retrieve.
pValue	[out] Pointer to NFDoubleCore that receives double core.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to double core count obtained using NFRecordGetDoubleCoreCount function.

See Also

NFRecord Module | HNFRecord | NFDoubleCore | NFRecordGetDoubleCoreCount | NFRecordSetDoubleCore

5.2.1.36. NFRecordGetDoubleCoreCapacity Function

Retrieves the number of double cores that the NFRecord can contain.

```
NResult N_API NFRecordGetDoubleCoreCapacity(
        HNFRecord hRecord,
        NInt * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NInt that receives number of double cores NFRecord can contain.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

Double core capacity is the number of double cores that the NFRecord can store. Double core count (see `NFRecordGetDoubleCoreCount` function) is the number of double cores that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding double cores the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | `NFRecordSetDoubleCoreCapacity` | `NFRecordGetDoubleCoreCount`

5.2.1.37. NFRecordGetDoubleCoreCount Function

Retrieves the number of double cores in the NFRecord.

```
NResult N_API NFRecordGetDoubleCoreCount(
        HNFRecord hRecord,
        NInt * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NInt that receives number of double cores.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

Double core capacity (see `NFRecordGetDoubleCoreCapacity` and `NFRecordSetDoubleCoreCapacity` functions) is the number of double cores that the NFRecord can store. Double core count is the number of double cores that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding

double cores the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | `NFRecordGetDoubleCoreCapacity` | `NFRecord-SetDoubleCoreCapacity`

5.2.1.38. NFRecordGetDoubleCores Function

Copies all double cores of NFRecord to the specified array.

```
NResult N_API NFRecordGetDoubleCores(
        HNFRecord hRecord,
        NFDoubleCore * arDoubleCores
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
arDoubleCores	[out] Pointer to array of NFDoubleCore that receives double cores.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *arDoubleCores* is NULL.

Remarks

Array *arDoubleCores* points to must be large enough to receive all NFRecord double cores. See `NFRecordGetDoubleCoreCount` function.

See Also

NFRecord Module | HNFRecord | NFDoubleCore | `NFRecordGetDoubleCoreCount`

5.2.1.39. NFRecordGetG Function

Retrieves the G of the NFRecord.


```
NResult N_API NFRecordGetG(
        HNFRecord hRecord,
        NByte * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NByte that receives G.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

G is a global fingerprint feature that reflects ridge density. It can have values from 0 to 255, so it occupies one byte. The bigger is value the bigger is ridge density.

See Also

NFRecord Module | HNFRecord | NFRecordSetG

5.2.1.40. NFRecordGetGMem Function

Retrieves the G of the packed NFRecord.

```
NResult N_API NFRecordGetGMem(
        const void * buffer,
        NSizeType bufferSize,
        NByte * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains

	packed NFRecord.
pValue	[out] Pointer to NByte that receives G.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

See Also

NFRecord Module

5.2.1.41. NFRecordGetHeight Function

Retrieves the height of the image the NFRecord is made from.

```
NResult N_API NFRecordGetHeight(
        HNFRecord hRecord,
        NUShort * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NUShort that receives height of fingerprint image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord

5.2.1.42. NFRecordGetHeightMem Function

Retrieves the height of the image the packed NFRecord is made from.

```
NResult N_API NFRecordGetHeightMem(
        const void * buffer,
        NSizeType bufferSize,
        NUShort * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.
pValue	[out] Pointer to NUShort that receives height of fingerprint image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

Always returns `1` for version 1.0 format.

See Also

NFRecord Module

5.2.1.43. NFRecordGetHorzResolution Function

Retrieves the horizontal resolution of the image the NFRecord is made from.

```
NResult N_API NFRecordGetHorzResolution(
        HNFRecord hRecord,
        NUShort * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NUShort that receives horizontal resolution in pixels per inch of fingerprint image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord

5.2.1.44. NFRecordGetHorzResolutionMem Function

Retrieves the horizontal resolution of the image the packed NFRecord is made from.

```
NResult N_API NFRecordGetHorzResolutionMem(
```



```
        const void * buffer,
        NSizeType bufferSize,
        NUShort * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.
pValue	[out] Pointer to NUShort that receives horizontal resolution in pixels per inch of fingerprint image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

Always returns `500` for version 1.0 format.

See Also

NFRecord Module

5.2.1.45. NFRecordGetImpressionType Function

Retrieves the impression type of the NFRecord.

```
NResult N_API NFRecordGetImpressionType(
        HNFRecord hRecord,
```

```
        NFImpressionType * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NFImpressionType that receives impression type.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord | NFImpressionType | NFRecordSetImpressionType

5.2.1.46. NFRecordGetImpressionTypeMem Function

Retrieves the impression type of the packed NFRecord.

```
NResult N_API NFRecordGetImpressionTypeMem(
        const void * buffer,
        NSizeType bufferSize,
        NFImpressionType * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.
pValue	[out] Pointer to NFImpressionType that receives impression type.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

Always returns `nfitLiveScanPlain` for version 1.0 format.

See Also

NFRecord Module | NFImpressionType

5.2.1.47. NFRecordGetMaxSize Function

Retrieves the maximal size of a packed NFRecord containing the specified number of minutiae, cores, deltas and double cores and the specified ridge counts type.

```
NResult N_API NFRecordGetMaxSize(
        NFMinutiaFormat minutiaFormat,
        NInt minutiaCount,
        NFRidgeCountsType ridgeCountsType,
        NInt coreCount,
        NInt deltaCount,
        NInt doubleCoreCount,
        NInt boWidth,
        NInt boHeight,
        NSizeType * pSize
);
```

Parameters

minutiaFormat	[in] The minutia format.
minutiaCount	[in] The number of minutiae.
ridgeCountsType	[in] The type of ridge counts.

coreCount	[in] The number of cores.
deltaCount	[in] The number of deltas.
doubleCoreCount	[in] The number of double cores.
boWidth	[in] The width of blocked orientations.
boHeight	[in] The height of blocked orientations.
pSize	[out] Pointer to NSizeType that receives maximal size of packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*minutiaFormat* is invalid. - or - *ridgeCountsType* is invalid.
N_E_ARGUMENT_NULL	*pSize* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*minutiaCount* is less than zero or greater than or equal to NFR_MAX_MINUTIA_COUNT. - or - *coreCount* is less than zero or greater than or equal to NFR_MAX_CORE_COUNT. - or - *deltaCount* is less than zero or greater than or equal to NFR_MAX_DELTA_COUNT. - or - *doubleCoreCount* is less than zero or greater than or equal to NFR_MAX_DOUBLE_CORE_COUNT.

Error Code	Condition
	- or - *boWidth* or *boHeight* is less than zero or greater than or equal to NFR_MAX_BLOCKED_ORIENTS_DIMENSION.

Remarks

This is a low-level function and can be changed in future version of the library.

The function calculates current (2.0) version packed size of NFRecord.

boWidth and *boHeight* parameters are for compatibility only. If one of them or both is zero, blocked orientations are ignored.

See Also

NFRecord Module | NFMinutiaFormat | NFMinutia | NFRidgeCountsType | NFCore | NFDelta | NFDoubleCore | NFRecordSaveToMemory

5.2.1.48. NFRecordGetMaxSizeV1 Function

Retrieves the maximal size of a packed in version 1.0 format NFRecord containing the specified number of minutiae, cores, deltas and double cores and the specified ridge counts type.

```
NResult N_API NFRecordGetMaxSizeV1(
        NFMinutiaFormat minutiaFormat,
        NInt minutiaCount,
        NInt coreCount,
        NInt deltaCount,
        NInt doubleCoreCount,
        NInt boWidth,
        NInt boHeight,
        NSizeType * pSize
);
```

Parameters

minutiaFormat	[in] The minutia format.
minutiaCount	[in] The number of minutiae.
coreCount	[in] The number of cores.
deltaCount	[in] The number of deltas.

doubleCoreCount	[in] The number of double cores.
boWidth	[in] The width of blocked orientations.
boHeight	[in] The height of blocked orientations.
pSize	[out] Pointer to NSizeType that receives maximal size of packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*minutiaFormat* is invalid.
N_E_ARGUMENT_NULL	*pSize* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*minutiaCount* is less than zero or greater than or equal to NFR_MAX_MINUTIA_COUNT. - or - *coreCount* is less than zero or greater than or equal to NFR_MAX_CORE_COUNT. - or - *deltaCount* is less than zero or greater than or equal to NFR_MAX_DELTA_COUNT. - or - *doubleCoreCount* is less than zero or greater than or equal to NFR_MAX_DOUBLE_CORE_COUNT. - or - *boWidth* or *boHeight* is less than zero or greater than or equal to NFR_MAX_BLOCKED_ORIENTS_DIMENSION.

Remarks

This is a low-level function and can be changed in future version of the library.

boWidth and *boHeight* parameters are for compatibility only. If one of them or both is zero, blocked orientations are ignored.

See Also

NFRecord Module | NFMinutiaFormat | NFMinutia | NFCore | NFDelta | NFDoubleCore | NFRecordSaveToMemoryV1

5.2.1.49. NFRecordGetMinutia Function

Retrieves the minutia at the specified index of the NFRecord.

```
NResult N_API NFRecordGetMinutia(
        HNFRecord hRecord,
        NInt index,
        NFMinutia * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of minutia to retrieve.
pValue	[out] Pointer to NFMinutia that receives minutia.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to minutia count obtained using NFRecordGetMinutiaCount function.

See Also

NFRecord Module | HNFRecord | NFMinutia | NFRecordGetMinutiaCount | NFRecordGetMinutiae

5.2.1.50. NFRecordGetMinutiaCapacity Function

Retrieves the number of minutiae that the NFRecord can contain.

```
NResult N_API NFRecordGetMinutiaCapacity(
        HNFRecord hRecord,
        NInt * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NInt that receives number of minutiae NFRecord can contain.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

Minutia capacity is the number of minutiae that the NFRecord can store. Minutia count (see NFRecordGetMinutiaCount function) is the number of minutiae that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding minutiae the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | NFRecordSetMinutiaCapacity | NFRecordGetMinutiaCount

5.2.1.51. NFRecordGetMinutiaCount Function

Retrieves the number of minutiae in the NFRecord.

```
NResult N_API NFRecordGetMinutiaCount(
        HNFRecord hRecord,
        NInt * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NInt that receives number of minutiae.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

Minutia capacity (see `NFRecordGetMinutiaCapacity` and `NFRecordSetMinutiaCapacity` functions) is the number of minutiae that the NFRecord can store. Minutia count is the number of minutiae that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding minutiae the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | `NFRecordGetMinutiaCapacity` | `NFRecordSetMinutiaCapacity`

5.2.1.52. NFRecordGetMinutiaFormat Function

Retrieves the format of the minutiae in NFRecord.

```
NResult N_API NFRecordGetMinutiaFormat(
        HNFRecord hRecord,
        NFMinutiaFormat * pValue
);
```


Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NFMinutiaFormat that receives format of minutiae in the NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord | NFMinutiaFormat | NFRecordSetMinutiaFormat

5.2.1.53. NFRecordGetMinutiaNeighbor Function

Retrieves the minutia neighbor at the specified index of the minutia at the specified index of the NFRecord.

```
NResult N_API NFRecordGetMinutiaNeighbor(
        HNFRecord hRecord,
        NInt minutiaIndex,
        NInt index,
        NFMinutiaNeighbor * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
minutiaIndex	[in] The index of minutia.
index	[in] Index of minutia neighbor to retrieve.
pValue	[out] Pointer to NFMinutiaNeighbor that receives minutia neighbor.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*minutiaIndex* is less than zero or greater than or equal to minutia count obtained using NFRecordGetMinutiaCount function. - or - *index* is less than zero or greater than or equal to minutia neighbor count obtained using NFRecordGetMinutiaNeighborCount function.

See Also

NFRecord Module | HNFRecord | NFMinutiaNeighbor | NFRecordGetMinutiaCount | NFRecordGetMinutiaNeighborCount | NFRecordSetMinutiaNeighbor

5.2.1.54. NFRecordGetMinutiaNeighborCount Function

Retrieves the number of minutia neighbors in the minutia at the specified index of the NFRecord.

```
NResult N_API NFRecordGetMinutiaNeighborCount(
        HNFRecord hRecord,
        NInt minutiaIndex,
        NInt * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
minutiaIndex	[in] The index of minutia.
pValue	[out] Pointer to NInt that receives number of minutia neighbors.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to minutia count obtained using NFRecordGetMinutiaCount function.

See Also

NFRecord Module | HNFRecord

5.2.1.55. NFRecordGetMinutiaNeighbors Function

Copies all minutia neighbors of the minutia at the specified index of the NFRecord to the specified array.

```
NResult N_API NFRecordGetMinutiaNeighbors(
        HNFRecord hRecord,
        NInt minutiaIndex,
        NFMinutiaNeighbor * arMinutiaNeighbors
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
minutiaIndex	[in] The index of minutia.
arMinutiaNeighbors	[out] Pointer to array of NFMinutiaNeighbor that receives minutia neighbors.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *arMinutiaNeighbors* is NULL.

Remarks

Array *arMinutiaNeighbors* points to must be large enough to receive all minutia neighbors. See NFRecordGetMinutiaNeighborCount function.

See Also

NFRecord Module | HNFRecord | NFMinutiaNeighbor | NFRecordGetMinutiaCount | NFRecordGetMinutiaNeighborCount

5.2.1.56. NFRecordGetMinutiae Function

Copies all minutiae of NFRecord to the specified array.

```
NResult N_API NFRecordGetMinutiae(
        HNFRecord hRecord,
        NFMinutia * arMinutiae
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
arMinutiae	[out] Pointer to array of NFMinutiathat receives minutiae.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *arMinutiae* is NULL.

Remarks

Array *arMinutiae* points to must be large enough to receive all NFRecord minutiae. See NFRecordGetMinutiaCount function.

See Also

NFRecord Module | HNFRecord | NFMinutia | NFRecordGetMinutiaCount

5.2.1.57. NFRecordGetPatternClass Function

Retrieves the pattern class of the NFRecord.

```
NResult N_API NFRecordGetPatternClass(
        HNFRecord hRecord,
        NFPatternClass * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NFPatternClass that receives fingerprint pattern class.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

Always returns `nfpcUnknown` for version 1.0 format.

See Also

NFRecord Module | HNFRecord | NFPatternClass | `NFRecordSetPatternClass`

5.2.1.58. NFRecordGetPatternClassMem Function

Retrieves the pattern class of the packed NFRecord.

```
NResult N_API NFRecordGetPatternClassMem(
        const void * buffer,
        NSizeType bufferSize,
        NFPatternClass * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.
pValue	[out] Pointer to NFPatternClass that receives fingerprint pattern class.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

Always returns `nfpcUnknown` for version 1.0 format.

See Also

NFRecord Module | NFPatternClass

5.2.1.59. NFRecordGetPosition Function

Retrieves the finger position of the NFRecord.

```
NResult N_API NFRecordGetPosition(
        HNFRecord hRecord,
        NFPosition * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

pValue	[out] Pointer to NFPosition that receives finger position.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord | NFPosition | NFRecordSetPosition

5.2.1.60. NFRecordGetPositionMem Function

Retrieves the finger position of the packed NFRecord.

```
NResult N_API NFRecordGetPositionMem(
        const void * buffer,
        NSizeType bufferSize,
        NFPosition * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.
pValue	[out] Pointer to NFPosition that receives finger position.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

Always returns `nfpUnknown` for version 1.0 format.

See Also

NFRecord Module | NFPosition

5.2.1.61. NFRecordGetQuality Function

Retrieves the quality of the NFRecord.

```
NResult N_API NFRecordGetQuality(
        HNFRecord hRecord,
        NByte * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NByte that receives fingerprint quality.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord | `NFRecordSetQuality`

5.2.1.62. NFRecordGetQualityMem Function

Retrieves the quality of the packed NFRecord.

```
NResult N_API NFRecordGetQualityMem(
        const void * buffer,
        NSizeType bufferSize,
        NByte * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.
pValue	[out] Pointer to NByte that receives finger-print quality.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

Always returns `0` for version 1.0 format.

See Also

NFRecord Module

5.2.1.63. NFRecordGetRidgeCountsType Function

Retrieves the ridge counts type the NFRecord contains.

```
NResult N_API NFRecordGetRidgeCountsType(
        HNFRecord hRecord,
        NFRidgeCountsType * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NFRidgeCountsType that receives ridge counts type stored in NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord | NFRidgeCountsType | NFRecordSetRidgeCountsType

5.2.1.64. NFRecordGetSize Function

Calculates packed size of the NFRecord.

```
NResult N_API NFRecordGetSize(
        HNFRecord hRecord,
        NUInt flags,
        NSizeType * pSize
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
flags	[in] Bitwise combination of zero or more flags that controls behavior of the function.
pSize	[out] Pointer to NSizeType that receives size of packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pSize* is NULL.

Remarks

The function calculates current (2.0) version packed size of NFRecord.

For the list of flags that are supported see `NFRecordSaveToMemory` function.

See Also

NFRecord Module | HNFRecord | `NFRecordSaveToMemory`

5.2.1.65. NFRecordGetSizeV1 Function

Calculates packed in version 1.0 format size of the NFRecord.

```
NResult N_API NFRecordGetSizeV1(
        HNFRecord hRecord,
        NUInt flags,
        NSizeType * pSize
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
flags	[in] Bitwise combination of zero or more flags that controls behavior of the function.
pSize	[out] Pointer to NSizeType that receives size of packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pSize* is NULL.

Remarks

For the list of flags that are supported see NFRecordSaveToMemoryV1 function.

See Also

NFRecord Module | HNFRecord | NFRecordSaveToMemoryV1

5.2.1.66. NFRecordGetVertResolution Function

Retrieves the vertical resolution of the image the NFRecord is made from.

```
NResult N_API NFRecordGetVertResolution(
        HNFRecord hRecord,
        NUShort * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NUShort that receives vertical resolution in pixels per inch of fingerprint image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord

5.2.1.67. NFRecordGetVertResolutionMem Function

Retrieves the vertical resolution of the image the packed NFRecord is made from.

```
NResult N_API NFRecordGetVertResolutionMem(
        const void * buffer,
        NSizeType bufferSize,
        NUShort * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.
pValue	[out] Pointer to NUShort that receives vertical resolution in pixels per inch of fingerprint image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

Always returns `500` for version 1.0 format.

See Also

NFRecord Module

5.2.1.68. NFRecordGetWidth Function

Retrieves the width of the image the NFRecord is made from.

```
NResult N_API NFRecordGetWidth(
        HNFRecord hRecord,
        NUShort * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
pValue	[out] Pointer to NUShort that receives width of fingerprint image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.

See Also

NFRecord Module | HNFRecord

5.2.1.69. NFRecordGetWidthMem Function

Retrieves the width of the image the packed NFRecord is made from.

```
NResult N_API NFRecordGetWidthMem(
        const void * buffer,
        NSizeType bufferSize,
        NUShort * pValue
);
```

Parameters

buffer	[in] Pointer to memory buffer that contains packed NFRecord.
bufferSize	[in] Size of memory buffer that contains packed NFRecord.
pValue	[out] Pointer to NUShort that receives width of fingerprint image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* or *pValue* is NULL.
N_E_END_OF_STREAM	*bufferSize* is less than expected.
N_E_FORMAT	Data in memory buffer *buffer* points to is inconsistent with NFRecord format.

Remarks

This function supports both NFRecord version 1.0 and 2.0 formats.

Always returns `1` for version 1.0 format.

See Also

NFRecord Module

5.2.1.70. NFRecordInsertCore Function

Inserts a NFCore into the NFRecord at the specified index.

```
NResult N_API NFRecordInsertCore(
        HNFRecord hRecord,
        NInt index,
        const NFCore * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

index	[in] Index at which core is inserted.
pValue	[in] Pointer to the NFCore to insert.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than core count obtained using NFRecordGetCoreCount function.
N_E_INVALID_OPERATION	Number of cores in NFRecord (see NFRecordGetCoreCount) is equal to NFR_MAX_CORE_COUNT.

See Also

NFRecord Module | HNFRecord | NFCore | NFRecordGetCoreCount

5.2.1.71. NFRecordInsertDelta Function

Inserts a NFDelta into the NFRecord at the specified index.

```
NResult N_API NFRecordInsertDelta(
        HNFRecord hRecord,
        NInt index,
        const NFDelta * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index at which delta is inserted.
pValue	[in] Pointer to the NFDelta to insert.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than delta count obtained using NFRecordGetDeltaCount function.
N_E_INVALID_OPERATION	Number of deltas in NFRecord (see NFRecordGetDeltaCount) is equal to NFR_MAX_DELTA_COUNT.

See Also

NFRecord Module | HNFRecord | NFDelta | NFRecordGetDeltaCount

5.2.1.72. NFRecordInsertDoubleCore Function

Inserts a NFDoubleCore into the NFRecord at the specified index.

```
NResult N_API NFRecordInsertDoubleCore(
        HNFRecord hRecord,
        NInt index,
        const NFDoubleCore * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index at which double core is inserted.
pValue	[in] Pointer to the NFDoubleCore to insert.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than double core count obtained using NFRecordGetDoubleCoreCount function.
N_E_INVALID_OPERATION	Number of double core in NFRecord (see NFRecordGetDoubleCoreCount) is equal to NFR_MAX_DOUBLE_CORE_COUNT.

See Also

NFRecord Module | HNFRecord | NFDoubleCore | NFRecordGetDoubleCoreCount

5.2.1.73. NFRecordInsertMinutia Function

Inserts a NFMinutia into the NFRecord at the specified index.

```
NResult N_API NFRecordInsertMinutia(
        HNFRecord hRecord,
        NInt index,
        const NFMinutia * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index at which minutia is inserted.
pValue	[in] Pointer to the NFMinutia to insert.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than minutia count obtained using NFRecordGetMinutiaCount function.
N_E_INVALID_OPERATION	Number of minutia in NFRecord (see NFRecordGetMinutiaCount) is equal to NFR_MAX_MINUTIA_COUNT.

See Also

NFRecord Module | HNFRecord | NFMinutia | NFRecordGetMinutiaCount

5.2.1.74. NFRecordRemoveCore Function

Removes the core at the specified index of the NFRecord.

```
NResult N_API NFRecordRemoveCore(
        HNFRecord hRecord,
        NInt index
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of core to remove.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to core count obtained using NFRecordGetCoreCount function.

See Also

NFRecord Module | HNFRecord | NFRecordGetCoreCount

5.2.1.75. NFRecordRemoveDelta Function

Removes the delta at the specified index of the NFRecord.

```
NResult N_API NFRecordRemoveDelta(
        HNFRecord hRecord,
        NInt index
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of delta to remove.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to delta count obtained using NFRecordGetDeltaCount function.

See Also

NFRecord Module | HNFRecord | NFRecordGetDeltaCount

5.2.1.76. NFRecordRemoveDoubleCore Function

Removes the double core at the specified index of the NFRecord.

```
NResult N_API NFRecordRemoveDoubleCore(
        HNFRecord hRecord,
        NInt index
);
```


Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of double core to remove.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to double core count obtained using NFRecordGetDoubleCoreCount function.

See Also

NFRecord Module | HNFRecord | NFRecordGetDoubleCoreCount

5.2.1.77. NFRecordRemoveMinutia Function

Removes the minutia at the specified index of the NFRecord.

```
NResult N_API NFRecordRemoveMinutia(
        HNFRecord hRecord,
        NInt index
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of minutia to remove.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to minutia count obtained using NFRecordGetMinutiaCount function.

See Also

NFRecord Module | HNFRecord | NFRecordGetMinutiaCount

5.2.1.78. NFRecordSaveToMemory Function

Packs the NFRecord into the specified memory buffer.

```
NResult N_API NFRecordSaveToMemory(
        HNFRecord hRecord,
        void * buffer,
        NSizeType bufferSize,
        NUInt flags,
        NSizeType * pSize
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
buffer	[out] Pointer to memory buffer to store packed NFRecord. Can be NULL.
bufferSize	[in] Size of memory buffer to store packed NFRecord.
flags	[in] Bitwise combination of zero or more flags that controls behavior of the function.
pSize	[out] Pointer to NSizeType that receives size of packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*buffer* is not NULL and *bufferSize* is less than size required to store packed NFRecord.
N_E_ARGUMENT_NULL	*hRecord* or *pSize* is NULL. - or - *buffer* is NULL and *bufferSize* is not zero.

Remarks

The function packs NFRecord in current (2.0) version format.

If *buffer* is NULL and *bufferSize* is zero the function only calculates the size of the buffer needed and has the same effect as NFRecordGetSize function.

If *buffer* is not NULL, *bufferSize* must not be less than value calculated with NFRecordGetSize function.

Note that blocked orientations are not packed by default.

The following flags are supported:

- NFR_SAVE_BLOCKED_ORIENTS
- NFR_SKIP_CURVATURES
- NFR_SKIP_GS
- NFR_SKIP_QUALITIES
- NFR_SKIP_RIDGE_COUNTS
- NFR_SKIP_SINGULAR_POINTS

See Also

NFRecord Module | HNFRecord | NFRecordGetSize | NFRecordCreateFromMemory

5.2.1.79. NFRecordSaveToMemoryV1 Function

Packs the NFRecord into the specified memory buffer in version 1.0 format.

```
NResult N_API NFRecordSaveToMemoryV1(
        HNFRecord hRecord,
        void * buffer,
        NSizeType bufferSize,
        NUInt flags,
```

```
        NSizeType * pSize
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
buffer	[out] Pointer to memory buffer to store packed NFRecord. Can be NULL.
bufferSize	[in] Size of memory buffer to store packed NFRecord.
flags	[in] Bitwise combination of zero or more flags that controls behavior of the function.
pSize	[out] Pointer to NSizeType that receives size of packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*buffer* is not NULL and *bufferSize* is less than size required to store packed NFRecord.
N_E_ARGUMENT_NULL	*hRecord* or *pSize* is NULL. - or - *buffer* is NULL and *bufferSize* is not zero.

Remarks

If *buffer* is NULL and *bufferSize* is zero the function only calculates the size of the buffer needed and has the same effect as NFRecordGetSizeV1 function.

If *buffer* is not NULL, *bufferSize* must not be less than value calculated with NFRecordGetSizeV1 function.

Note that blocked orientations are not packed by default.

The following flags are supported:

- NFR_SAVE_BLOCKED_ORIENTS
- NFR_SKIP_CURVATURES
- NFR_SKIP_GS
- NFR_SKIP_SINGULAR_POINTS

See Also

NFRecord Module | HNFRecord | `NFRecordCreateFromMemory` | `NFRecordGetSizeV1`

5.2.1.80. NFRecordSetCbeffProductType Function

Sets the Cbeff product type.

```
NResult N_API NFRecordSetCbeffProductType(
    HNFRecord hRecord,
    NUShort value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] Cbeff product type.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord | NFRecordGetCbeffProductType NFRecordGetCbeffProductTypeMem

5.2.1.81. NFRecordSetCore Function

Sets a NFCore at the specified index of the NFRecord.


```
NResult N_API NFRecordSetCore(
        HNFRecord hRecord,
        NInt index,
        const NFCore * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of core to set.
pValue	[in] Pointer to the NFCore to set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to core count obtained using NFRecordGetCoreCount function.

See Also

NFRecord Module | HNFRecord | NFCore | NFRecordGetCoreCount | NFRecordGetCore

5.2.1.82. NFRecordSetCoreCapacity Function

Sets the number of cores that the NFRecord can contain.

```
NResult N_API NFRecordSetCoreCapacity(
        HNFRecord hRecord,
        NInt value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New number of cores NFRecord can contain.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*value* is less than core count obtained using NFRecordGetCoreCount function.

Remarks

Core capacity is the number of cores that the NFRecord can store. Core count (see NFRecordGetCoreCount function) is the number of cores that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding cores the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | NFRecordGetCoreCapacity | NFRecordGetCoreCount

5.2.1.83. NFRecordSetDelta Function

Sets a NFDelta at the specified index of the NFRecord.

```
NResult N_API NFRecordSetDelta(
        HNFRecord hRecord,
        NInt index,
        const NFDelta * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

index	[in] Index of delta to set.
pValue	[in] Pointer to the NFDelta to set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to delta count obtained using NFRecordGetDeltaCount function.

See Also

NFRecord Module | HNFRecord | NFDelta | NFRecordGetDeltaCount | NFRecordGetDelta

5.2.1.84. NFRecordSetDeltaCapacity Function

Sets the number of deltas that the NFRecord can contain.

```
NResult N_API NFRecordSetDeltaCapacity(
        HNFRecord hRecord,
        NInt value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New number of deltas NFRecord can contain.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*value* is less than delta count obtained using NFRecordGetDeltaCount function.

Remarks

Delta capacity is the number of deltas that the NFRecord can store. Delta count (see NFRecordGetDeltaCount function) is the number of deltas that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding deltas the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | NFRecordGetDeltaCapacity | NFRecordGetDeltaCount

5.2.1.85. NFRecordSetDoubleCore Function

Sets a NFDoubleCore at the specified index of the NFRecord.

```
NResult N_API NFRecordSetDoubleCore(
        HNFRecord hRecord,
        NInt index,
        const NFDoubleCore * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
index	[in] Index of double core to set.
pValue	[in] Pointer to the NFDoubleCore to set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to double core count obtained using NFRecordGetDoubleCoreCount function.

See Also

NFRecord Module | HNFRecord | NFDoubleCore | NFRecordGetDoubleCoreCount | NFRecordGetDoubleCore

5.2.1.86. NFRecordSetDoubleCoreCapacity Function

Sets the number of double cores that the NFRecord can contain.

```
NResult N_API NFRecordSetDoubleCoreCapacity(
        HNFRecord hRecord,
        NInt value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New number of double cores NFRecord can contain.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*value* is less than double core count obtained using NFRecordGetDoubleCoreCount function.

Remarks

Double core capacity is the number of double cores that the NFRecord can store. Double core count (see `NFRecordGetDoubleCoreCount` function) is the number of double cores that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding double cores the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | `NFRecordGetDoubleCoreCapacity` | `NFRecordGetDoubleCoreCount`

5.2.1.87. NFRecordSetG Function

Sets the G of the NFRecord.

```
NResult N_API NFRecordSetG(
        HNFRecord hRecord,
        NByte value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New G value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.

Remarks

G is a global fingerprint feature that reflects ridge density. It can have values from 0 to 255, so it occupies one byte. The bigger is value the bigger is ridge density.

See Also

NFRecord Module | HNFRecord | NFRecordGetG

5.2.1.88. NFRecordSetImpressionType Function

Sets the impression type of the NFRecord.

```
NResult N_API NFRecordSetImpressionType(
        HNFRecord hRecord,
        NFImpressionType value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New impression type value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*value* is invalid.
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord | NFImpressionType | NFRecordGetImpressionType

5.2.1.89. NFRecordSetMinutia Function

Sets a NFMinutia at the specified index of the NFRecord.

```
NResult N_API NFRecordSetMinutia(
        HNFRecord hRecord,
        NInt index,
        const NFMinutia * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

index	[in] Index of minutia to set.
pValue	[in] Pointer to the NFMinutia to set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to minutia count obtained using NFRecordGetMinutiaCount function.

See Also

NFRecord Module | HNFRecord | NFMinutia | NFRecordGetMinutiaCount | NFRecordGetMinutia

5.2.1.90. NFRecordSetMinutiaCapacity Function

Sets the number of minutiae that the NFRecord can contain.

```
NResult N_API NFRecordSetMinutiaCapacity(
        HNFRecord hRecord,
        NInt value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New number of minutiae NFRecord can contain.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*value* is less than minutia count obtained using NFRecordGetMinutiaCount function.

Remarks

Minutia capacity is the number of minutiae that the NFRecord can store. Minutia count (see NFRecordGetMinutiaCount function) is the number of minutiae that are actually in the NFRecord.

Capacity is always greater than or equal to count. If count exceeds capacity while adding minutiae the capacity is automatically increased by reallocating the internal array before copying the old elements and adding the new elements.

See Also

NFRecord Module | HNFRecord | NFRecordGetMinutiaCapacity | NFRecordGetMinutiaCount

5.2.1.91. NFRecordSetMinutiaFormat Function

Sets the format of the minutiae in NFRecord.

```
NResult N_API NFRecordSetMinutiaFormat(
        HNFRecord hRecord,
        NFMinutiaFormat value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New minutia format value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*value* is invalid.
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord | NFMinutiaFormat | NFRecordGetMinutiaFormat

5.2.1.92. NFRecordSetMinutiaNeighbor Function

Sets a NFMinutiaNeighbor at the specified index of the minutia at the specified index of the NFRecord.

```
NResult N_API NFRecordSetMinutiaNeighbor(
        HNFRecord hRecord,
        NInt minutiaIndex,
        NInt index,
        const NFMinutiaNeighbor * pValue
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
minutiaIndex	[in] The index of minutia.
index	[in] Index of minutia neighbor to set.
pValue	[in] Pointer to the NFMinutiaNeighbor to set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*hRecord* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*minutiaIndex* is less than zero or greater than or equal to minutia count obtained us-

Error Code	Condition
	ing NFRecordGetMinutiaCount function. - or - *index* is less than zero or greater than or equal to minutia neighbor count obtained using NFRecordGetMinutiaNeighborCount function.

See Also

NFRecord Module | HNFRecord | NFMinutiaNeighbor | NFRecordGetMinutiaCount | NFRecordGetMinutiaNeighborCount | NFRecordGetMinutiaNeighbor

5.2.1.93. NFRecordSetPatternClass Function

Sets the pattern class of the NFRecord.

```
NResult N_API NFRecordSetPatternClass(
        HNFRecord hRecord,
        NFPatternClass value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New fingerprint pattern class value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*value* is invalid.
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord | NFPatternClass | NFRecordGetPatternClass

5.2.1.94. NFRecordSetPosition Function

Sets the finger position of the NFRecord.

```
NResult N_API NFRecordSetPosition(
        HNFRecord hRecord,
        NFPosition value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New finger position value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*value* is invalid.
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord | NFPosition | NFRecordGetPosition

5.2.1.95. NFRecordSetQuality Function

Sets the quality of the NFRecord.

```
NResult N_API NFRecordSetQuality(
        HNFRecord hRecord,
        NByte value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.

value	[in] New fingerprint quality value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord | NFRecordGetQuality

5.2.1.96. NFRecordSetRidgeCountsType Function

Sets the ridge counts type the NFRecord contains.

```
NResult N_API NFRecordSetRidgeCountsType(
        HNFRecord hRecord,
        NFRidgeCountsType value
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
value	[in] New ridge counts type value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*value* is invalid.
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord | NFRidgeCountsType | `NFRecordGetRidgeCount-sType`

5.2.1.97. NFRecordSortMinutiae Function

Sorts minutiae in NFRecord by the specified order.

```
NResult N_API NFRecordSortMinutiae(
        HNFRecord hRecord,
        NFMinutiaOrder order
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
order	[in] Specifies minutia order.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *order* is incorrect.
N_E_ARGUMENT_NULL	*hRecord* is NULL.

See Also

NFRecord Module | HNFRecord | NFMinutiaOrder

5.2.1.98. NFRecordTruncateMinutiae Function

Truncates minutiae in NFRecord by peeling off the minutiae convex hull while minutia count is greater than the specified maximal count.

```
NResult N_API NFRecordTruncateMinutiae(
        HNFRecord hRecord,
        NInt maxCount
);
```


Parameters

hRecord	[in] Handle to the NFRecord object.
maxCount	Maximal minutia count to be present in the NFRecord after truncation.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*maxCount* is less than zero or greater than NFR_MAX_MINUTIA_COUNT.

See Also

NFRecord Module | HNFRecord | NFRecordAddMinutia

5.2.1.99. NFRecordTruncateMinutiaeByQuality Function

Truncates minutiae in NFRecord by removing minutiae which NFMinutia.Quality field value is less than the specified threshold while minutia count is greater than the specified maximal count.

```
NResult N_API NFRecordTruncateMinutiaeByQuality(
        HNFRecord hRecord,
        NByte threshold,
        NInt maxCount
);
```

Parameters

hRecord	[in] Handle to the NFRecord object.
threshold	Specifies minimal NFMinutia.Quality field value of minutiae not to be removed.
maxCount	Maximal minutia count to be present in the NFRecord after truncation.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_OUT_OF_RANGE	*maxCount* is less than zero or greater than NFR_MAX_MINUTIA_COUNT.
N_E_ARGUMENT_NULL	*hRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*threshold* is incorrect.
N_E_INVALID_OPERATION	Minutia format is not NFMinutia-Format.nfmfHasQuality.

Remarks

Minutia count after truncation may be greater than *maxCount* if there is no enough minutiae with quality less than *threshold*.

See Also

NFRecord Module | HNFRecord | NFRecordAddMinutia

5.2.1.100. NFRidgeCountsType Enumeration

Specifies the type of ridge counts contained in a NFRecord.

```
typedef enum NFRidgeCountsType_ { } NFRidgeCountsType;
```

Members

nfrctEightNeighbors	The NFRecord contains ridge counts to closest minutia in each of the eight sectors of each minutia. First sector starts at minutia angle.
nfrctEightNeighborsWith-Indexes	The NFRecord contains ridge counts to eight neighbors of each minutia.
nfrctFourNeighbors	The NFRecord contains ridge counts to closest minutia in each of the four sectors of each minutia. First sector starts at minutia angle.

`nfrctFourNeighborsWithIndexes`	The NFRecord contains ridge counts to four neighbors of each minutia.
`nfrctNone`	The NFRecord does not contain ridge counts.
`nfrctUnspecified`	For internal use.

Remarks

Extracted template with nfrctEightNeighborsWithIndexes parameter is bigger than the template extracted with nfrctEightNeighbors parameter. Templates extracted with nfrctEightNeighborsWithIndexes parameter is faster than the templates extracted with nfrctEightNeighbors parameter.

Extracted template with nfrctFourNeighborsWithIndexes parameter is bigger than the template extracted with nfrctFourNeighbors parameter. Templates extracted with nfrctFourNeighborsWithIndexes parameter is faster than the templates extracted with nfrctFourNeighbors parameter.

See Also

NFRecord Module

5.3. NImages Library

Provides functionality for loading, saving and converting images in various formats.

Windows

Import library: `NImages.dll.lib`.

DLL: `NImages.dll`.

Requirements:

- `NCore.dll`.

Linux

Shared object: `libNImages.so`.

Requirements:

- `libNCore.so`.

Modules

Bmp	Provides functionality for loading and saving images in BMP format.
Jpeg	Provides functionality for loading and saving images in JPEG format.
NGrayscaleImage	Provides functionality for managing 8-bit grayscale images.
NImageFormat	Provides functionality for loading and saving images in format-neutral way.
NImage	Provides functionality for managing images.
NImages	Provides library registration and other additional functionality.
NMonochromeImage	Provides functionality for managing 1-bit monochrome images.
NPixelFormat	Provides functionality for working with image pixel format.
NRgbImage	Provides functionality for managing 24-bit RGB images.
Tiff	Provides functionality for loading images in TIFF format.

5.3.1. Bmp Module

Provides functionality for loading and saving images in BMP format.

Header file: `Bmp.h`.

Functions

`BmpLoadImageFromFile`	Loads image from BMP file.
`BmpLoadImageFromHBitmap`	Loads image from Windows HBITMAP.
`BmpLoadImageFromMemory`	Loads image from memory buffer containing BMP file.
`BmpSaveImageToFile`	Saves image to file in BMP format.
`BmpSaveImageToHBitmap`	Saves image to Windows HBITMAP.
`BmpSaveImageToMemory`	Saves image to memory buffer in BMP

	format.

See Also

NImages Library

5.3.1.1. BmpLoadImageFromFile Function

Loads image from BMP file.

```
NResult N_API BmpLoadImageFromFile(
        const NChar * szFileName,
        HNImage * pHImage
);
```

Parameters

szFileName	[in] Points to string that specifies file name.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | BmpLoadImageFromMemory | BmpLoadImageFromHBitmap | BmpSaveImageToFile

5.3.1.2. BmpLoadImageFromHBitmap Function

Note
This function is available only on Windows.

Loads image from Windows HBITMAP.

```
NResult N_API BmpLoadImageFromHBitmap(
        NHandle handle,
        HNImage * pHImage
);
```

Parameters

handle	[in] Handle that specifies Windows HBITMAP.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*handle* or *pHImage* is NULL.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | `BmpLoadImageFromFile` | `BmpLoadImageFromMemory` | `BmpSaveImageToHBitmap`

5.3.1.3. BmpLoadImageFromMemory Function

Loads image from memory buffer containing BMP file.

```
NResult N_API BmpLoadImageFromMemory(
        const void * buffer,
        NSizeType bufferLength,
```



```
        HNImage * pHImage
);
```

Parameters

buffer	[in] Pointer to memory buffer.
bufferLength	[in] Length of memory buffer.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* is NULL and *bufferLength* is not equal to zero. - or - *pHImage* is NULL.
N_E_FORMAT	Format of file contained in buffer specified by *buffer* is invalid.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | `BmpLoadImageFromFile` | `BmpLoadImageFromHBitmap` | `BmpSaveImageToMemory`

5.3.1.4. BmpSaveImageToFile Function

Saves image to file in BMP format.

```
NResult N_API BmpSaveImageToFile(
        HNImage hImage,
        const NChar * szFileName
```

```
);
```

Parameters

hImage	[in] Handle to image.
szFileName	[in] Points to string that specifies file name.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *szFileName* is NULL.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | `BmpSaveImageToMemory` | `BmpSaveImageToHBitmap` | `BmpLoadImageFromFile`

5.3.1.5. BmpSaveImageToHBitmap Function

> **Note**
> This function is available only on Windows.

Saves image to Windows HBITMAP.

```
NResult N_API BmpSaveImageToHBitmap(
        HNImage hImage,
        NHandle * pHandle
);
```

Parameters

hImage	[in] Handle to image.
pHandle	[out] Pointer to NHandle that receives handle to created Windows HBITMAP.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pHandle* is NULL.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Bmp Module | HNImage | BmpSaveImageToFile | BmpSaveImageToMemory Bmp-LoadImageFromHBitmap

5.3.1.6. BmpSaveImageToMemory Function

Saves image to memory buffer in BMP format.

```
NResult N_API BmpSaveImageToMemory(
        HNImage hImage,
        void * * pBuffer,
        NSizeType * pBufferLength
);
```

Parameters

hImage	[in] Handle to image.
pBuffer	[out] Pointer to void * that receives pointer to allocated memory buffer.
pBufferLength	[out] Pointer to NSizeType that receives size of allocated memory buffer.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage*, *pBuffer* or *pBufferLength* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory to allocate memory buffer.

Remarks

This is a low-level function and can be changed in future version of the library.

Memory buffer allocated by the function must be deallocated using NFree function when it is no longer needed.

See Also

Bmp Module | HNImage | BmpSaveImageToFile | BmpSaveImageToHBitmap | BmpLoadImageFromMemory

5.3.2. Jpeg Module

Provides functionality for loading and saving images in JPEG format.

Header file: Jpeg.h.

Functions

JpegLoadImageFromFile	Loads image from JPEG file.
JpegLoadImageFromMemory	Loads image from memory buffer containing JPEG file.
JpegSaveImageToFile	Saves image to file in JPEG format with specified bitrate.
JpegSaveImageToMemory	Saves image to memory buffer in JPEG format with specified bitrate.

Macros

JPEG_DEFAULT_QUALITY	Specifies default JPEG quality.

See Also

NImages Library

5.3.2.1. JpegLoadImageFromFile Function

Loads image from JPEG file.

```
NResult N_API JpegLoadImageFromFile(
        const NChar * szFileName,
        HNImage * pHImage
);
```

Parameters

szFileName	[in] Points to string that specifies file name.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid.

See Also

Jpeg Module | HNImage | `JpegLoadImageFromMemory` | `JpegSaveImageToFile`

5.3.2.2. JpegLoadImageFromMemory Function

Loads image from memory buffer containing JPEG file.

```
NResult N_API JpegLoadImageFromMemory(
        const void * buffer,
        NSizeType bufferLength,
        HNImage * pHImage
);
```

Parameters

buffer	[in] Pointer to memory buffer.
bufferLength	[in] Length of memory buffer.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* is NULL and *bufferLength* is not equal to zero. - or - *pHImage* is NULL.
N_E_FORMAT	Format of file contained in buffer specified by *buffer* is invalid.

See Also

Jpeg Module | HNImage | JpegLoadImageFromFile | JpegSaveImageToMemory

5.3.2.3. JpegSaveImageToFile Function

Saves image to file in JPEG format with specified bitrate.

```
NResult N_API JpegSaveImageToFile(
        HNImage hImage,
        NInt quality,
        const NChar * szFileName
);
```

Parameters

hImage	[in] Handle to image.
quality	[in] Specifies quality of JPEG image.
szFileName	[in] Points to string that specifies file name.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *szFileName* is NULL.

See Also

Jpeg Module | HNImage | JPEG_DEFAULT_QUALITY | JpegSaveImageToMemory | JpegLoadImageFromFile

5.3.2.4. JpegSaveImageToMemory Function

Saves image to memory buffer in JPEG format with specified bitrate.

```
NResult N_API JpegSaveImageToMemory(
        HNImage hImage,
        NInt quality,
        void * * pBuffer,
        NSizeType * pBufferLength
);
```

Parameters

hImage	[in] Handle to image.
quality	[in] Specifies quality of JPEG image.
pBuffer	[out] Pointer to void * that receives pointer to allocated memory buffer.
pBufferLength	[out] Pointer to NSizeType that receives size of allocated memory buffer.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage*, *pBuffer* or *pBufferLength*

Error Code	Condition
	is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory to allocate memory buffer.

Remarks

Memory buffer allocated by the function must be deallocated using `NFree` function when it is no longer needed.

See Also

Jpeg Module | HNImage | JPEG_DEFAULT_QUALITY | `JpegSaveImageToFile` | `JpegLoadImageFromMemory`

5.3.3. NGrayscaleImage Module

Provides functionality for managing 8-bit grayscale images.

Header file: `NGrayscaleImage.h`.

Functions

`NGrayscaleImageGetPixel`	Retrieves value of pixel at the specified co-ordinates in 8-bit grayscale image.
`NGrayscaleImageSetPixel`	Sets value of pixel at the specified coordinates in 8-bit grayscale image.

Remarks

This module provides advanced functionality, such as individual pixel value retrieval for image with pixel format equal to `npfGrayscale`.

See Also

NImages Library | NImage Module

5.3.3.1. NGrayscaleImageGetPixel Function

Retrieves value of pixel at the specified coordinates in 8-bit grayscale image.

```
NResult N_API NGrayscaleImageGetPixel(
        HNImage hImage,
```

```
        NUInt x,
        NUInt y,
        NByte * pValue
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
pValue	[out] Points to NByte that receives pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to `npfGrayscale`.

See Also

NGrayscaleImage Module | HNImage | `NGrayscaleImageSetPixel`

5.3.3.2. NGrayscaleImageSetPixel Function

Sets value of pixel at the specified coordinates in 8-bit grayscale image.

```
NResult N_API NGrayscaleImageSetPixel(
        HNImage hImage,
        NUInt x,
```

```
        NUInt y,
        NByte value
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
value	[in] Specifies new pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to `npf-Grayscale`.

See Also

NGrayscaleImage Module | HNImage | `NGrayscaleImageGetPixel`

5.3.4. NImageFormat Module

Provides functionality for loading and saving images in format-neutral way.

Header file: `NImageFormat.h`.

Functions

NImageFormatCanRead	Retrieves a value indicating whether the image format supports reading.
NImageFormatCanWrite	Retrieves a value indicating whether the image format supports writing.
NImageFormatGetBmp	Retrieves BMP image format.
NImageFormatGetDefaultFileExtension	Retrieves default file extension of the image format.
NImageFormatGetFileFilter	Retrieves file filter of the image format.
NImageFormatGetFormat	Retrieves supported image format with specified index.
NImageFormatGetFormatCount	Retrieves number of supported image formats.
NImageFormatGetName	Retrieves name of the image format.
NImageFormatGetTiff	Retrieves TIFF image format.
NImageFormatLoadImageFromFile	Loads image from file of specified image format.
NImageFormatLoadImageFromMemory	Loads image from the memory buffer containing file of specified image format.
NImageFormatSaveImageToFile	Saves image to the file in specified format.
NImageFormatSaveImageToMemory	Saves image to the memory buffer in specified format.
NImageFormatSelect	Retrieves supported image format registered with file extension of specified file name and supporting reading/writing as specified.

Types

HNImageFormat	Handle to image format.

See Also

NImages Library | NImage Module

5.3.4.1. NImageFormatCanRead Function

Retrieves a value indicating whether the image format supports reading.

```
NResult N_API NImageFormatCanRead(
        HNImageFormat hImageFormat,
        NBool * pValue
);
```

Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to NBool that receives value indicating whether the image format supports reading.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* or *pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatCanWrite`

5.3.4.2. NImageFormatCanWrite Function

Retrieves a value indicating whether the image format supports writing.

```
NResult N_API NImageFormatCanWrite(
        HNImageFormat hImageFormat,
        NBool * pValue
);
```

Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to NBool that receives value indicating whether the image format supports writing.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* or *pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | NImageFormatCanRead

5.3.4.3. NImageFormatGetBmp Function

Retrieves BMP image format.

```
NResult N_API NImageFormatGetBmp(
        HNImageFormat * pValue
);
```

Parameters

pValue	[out] Pointer to HNImageFormat that receives handle to image format.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | NImageFormatGetTiff

5.3.4.4. NImageFormatGetDefaultFileExtension Function

Retrieves default file extension of the image format.

```
NResult N_API NImageFormatGetDefaultFileExtension(
        HNImageFormat hImageFormat,
        NChar * pValue
);
```

Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to string that receives default file extension of the image format. Can be NULL.

Return Values

If the function succeeds and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* is NULL.

See Also

NImageFormat Module | HNImageFormat

5.3.4.5. NImageFormatGetFileFilter Function

Retrieves file filter of the image format.

```
NResult N_API NImageFormatGetFileFilter(
        HNImageFormat hImageFormat,
        NChar * pValue
);
```

Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to string that receives file filter of the image format. Can be NULL.

Return Values

If the function succeeds and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* is NULL.

See Also

NImageFormat Module | HNImageFormat

5.3.4.6. NImageFormatGetFormat Function

Retrieves supported image format with specified index.

```
NResult N_API NImageFormatGetFormat(
        NInt index,
        HNImageFormat * pValue
);
```

Parameters

index	[in] Specifies zero-based supported image format index to retrieve.
pValue	[out] Pointer to NImageFormat that receives image format.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*index* is less than zero or greater than or equal to supported image format count. See

Error Code	Condition
	NImageFormatGetFormatCount.

See Also

NImageFormat Module | HNImageFormat | NImageFormatGetFormatCount

5.3.4.7. NImageFormatGetFormatCount Function

Retrieves number of supported image formats.

```
NResult N_API NImageFormatGetFormatCount(
        NInt * pValue
);
```

Parameters

pValue	[out] Pointer to NInt that receives number of supported image formats.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | NImageFormatGetFormat

5.3.4.8. NImageFormatGetName Function

Retrieves name of the image format.

```
NResult N_API NImageFormatGetName(
        HNImageFormat hImageFormat,
        NChar * pValue
);
```


Parameters

hImageFormat	[in] Handle to image format.
pValue	[out] Pointer to string that receives name of the image format. Can be NULL.

Return Values

If the function succeeds and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* is NULL.

See Also

NImageFormat Module | HNImageFormat

5.3.4.9. NImageFormatGetTiff Function

Retrieves TIFF image format.

```
NResult N_API NImageFormatGetTiff(
        HNImageFormat * pValue
);
```

Parameters

pValue	[out] Pointer to HNImageFormat that receives handle to image format.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatGetBmp`

5.3.4.10. NImageFormatLoadImageFromFile Function

Loads image from file of specified image format.

```
NResult N_API NImageFormatLoadImageFromFile(
        HNImageFormat hImageFormat,
        const NChar * szFileName,
        HNImage * pHImage
);
```

Parameters

hImageFormat	[in] Handle to image format.
szFileName	[in] Points to string that specifies file name.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat*, *szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid for specified image format.
N_E_NOT_SUPPORTED	Image format specified by *hImageFormat* does not support reading.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatCanRead` | HNImage | `NImageFormatLoadImageFromMemory` | `NImageFormatSaveImageToFile`

5.3.4.11. NImageFormatLoadImageFromMemory Function

Loads image from the memory buffer containing file of specified image format.

```
NResult N_API NImageFormatLoadImageFromMemory(
        HNImageFormat hImageFormat,
        void * buffer,
        NSizeType bufferLength,
        HNImage * pHImage
);
```

Parameters

hImageFormat	[in] Handle to image format.
buffer	[in] Pointer to memory buffer.
bufferLength	[in] Length of memory buffer.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat* or *pHImage* is NULL. - or - *buffer* is NULL and *bufferLength* is not equal to zero.
N_E_FORMAT	Format of file contained in buffer specified by *buffer* is invalid for specified image format.
N_E_NOT_SUPPORTED	Image format specified by *hImageFormat* does not support reading.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatCanRead` | HNImage | `NImageFormatLoadImageFromFile` | `NImageFormatSaveImageToMemory`

5.3.4.12. NImageFormatSaveImageToFile Function

Saves image to the file in specified format.

```
NResult N_API NImageFormatSaveImageToFile(
        HNImageFormat hImageFormat,
        HNImage hImage,
        const NChar * szFileName
);
```

Parameters

hImageFormat	[in] Handle to image format.
hImage	[in] Handle to image.
szFileName	[in] Points to string that specifies file name.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat*, *hImage* or *szFileName* is NULL.
N_E_NOT_SUPPORTED	Image format specified by *hImageFormat* does not support writing.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatCanWrite` | HNImage | `NImageFormatSaveImageToMemory` | `NImageFormatLoadImageFromFile`

5.3.4.13. NImageFormatSaveImageToMemory Function

Saves image to the memory buffer in specified format.


```
NResult N_API NImageFormatSaveImageToMemory(
        HNImageFormat hImageFormat,
        HNImage hImage,
        void * * pBuffer,
        NSizeType * pBufferLength
);
```

Parameters

hImageFormat	[in] Handle to image format.
hImage	[in] Handle to image.
pBuffer	[out] Pointer to void * that receives pointer to allocated memory buffer.
pBufferLength	[out] Pointer to NSizeType that receives size of allocated memory buffer.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImageFormat*, *hImage*, *pBuffer* or *pBufferLength* is NULL.
N_E_NOT_SUPPORTED	Image format specified by *hImage-Format* does not support writing.

Remarks

Memory buffer allocated by the function must be deallocated using NFree function when it is no longer needed.

See Also

NImageFormat Module | HNImageFormat | NImageFormatCanWrite | HNImage | NImageFormatSaveImageToFile | NImageFormatLoadImageFromMemory

5.3.4.14. NImageFormatSelect Function

Retrieves supported image format registered with file extension of specified file name and

supporting reading/writing as specified.

```
NResult N_API NImageFormatSelect(
        const NChar * szFileName,
        NFileAccess fileAccess,
        HNImageFormat * pHImageFormat
);
```

Parameters

szFileName	[in] Points to string that file name.
fileAccess	[in] Specifies that image format should support reading, writing or both.
pHImageFormat	[out] Pointer to HNImageFormat that receives handle to image format.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*fileAccess* value is invalid.
N_E_ARGUMENT_NULL	*szFileName* or *pHImageFormat* is NULL.

Remarks

If none of supported image formats that supports reading/writing as specified by *fileAccess* is registered with file extension of *szFileName* then handle returned via *pHImageFormat* is NULL.

See Also

NImageFormat Module | HNImageFormat | NFileAccess | `NImageFormatGetFormatCount` | `NImageFormatGetFormat`

5.3.5. NImage Module

Provides functionality for managing images.

Header file: `NImage.h`.

Functions

`NImageClone`	Creates a new image that is a copy of specified image.
`NImageCreate`	Creates an image with specified pixel format, size, stride and resolution.
`NImageCreateFromData`	Creates an image with specified pixel format, size, stride and resolution and copies specified pixels to it.
`NImageCreateFromFile`	Creates (loads) an image from file of specified format.
`NImageCreateFromImage`	Creates an image from specified image with specified pixel format and stride.
`NImageCreateFromImageEx`	Creates an image from specified image with specified pixel format, stride and resolution.
`NImageCreateWrapper`	Creates an image wrapper for specified pixels with specified pixel format, size, stride and resolution.
`NImageFree`	Deletes the image. After the image is deleted the specified handle is no longer valid.
`NImageGetHeight`	Retrieves height of the image.
`NImageGetHorzResolution`	Retrieves horizontal resolution of the image.
`NImageGetPixelFormat`	Retrieves pixel format of the image.
`NImageGetPixels`	Retrieves pointer to memory block containing pixels of the image.
`NImageGetSize`	Retrieves size of memory block containing pixels of the image.
`NImageGetStride`	Retrieves stride (size of one row) of the image.
`NImageGetVertResolution`	Retrieves vertical resolution of the image.
`NImageGetWidth`	Retrieves width of the image.
`NImageSaveToFile`	Saves the image to the file of specified format.

Types

HNImage	Handle to image.

See Also

NImages Library | NMonochromeImage Module | NGrayscaleImage Module | NRgbImage Module | NImageFormat Module

5.3.5.1. NImageClone Function

Creates a new image that is a copy of specified image.

```
NResult N_API NImageClone(
        HNImage hImage,
        HNImage * pHClonedImage
);
```

Parameters

hImage	[in] Handle to the image.
pHClonedImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pHClonedImage* is NULL.

Remarks

Created image must be deleted using `NImageFree` function.

See Also

NImage Module | HNImage | `NImageFree` | `NImageCreate`

5.3.5.2. NImageCreate Function

Creates an image with specified pixel format, size, stride and resolution.

```
NResult N_API NImageCreate(
        NPixelFormat pixelFormat,
        NUInt width,
        NUInt height,
        NSizeType stride,
        NFloat horzResolution,
        NFloat vertResolution,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
width	[in] Specifies width of the image.
height	[in] Specifies height of the image.
stride	[in] Specifies stride of the image. Can be zero.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is not zero and is less than minimal value for specified pixel format and width.
N_E_ARGUMENT_NULL	*pHImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero.

Error Code	Condition
	- or - *horzResolution* or *vertResolution* is less than zero.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride function.

Created image must be deleted using NImageFree function.

horzResolution and *vertResolution* can be zero if resolution is not applicable for the image.

See Also

NImage Module | HNImage | NImageFree | NImageCreateWrapper | NImageCreateFromData | NImageCreateFromImage | NImageCreateFromFile | NImageClone | NImageGetStride

5.3.5.3. NImageCreateFromData Function

Creates an image with specified pixel format, size, stride and resolution and copies specified pixels to it.

```
NResult N_API NImageCreateFromData(
        NPixelFormat pixelFormat,
        NUInt width,
        NUInt height,
        NSizeType stride,
        NFloat horzResolution,
        NFloat vertResolution,
        NSizeType srcStride,
        const void * srcPixels,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
width	[in] Specifies width of the image.
height	[in] Specifies height of the image.

stride	[in] Specifies stride of the image. Can be zero.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
srcStride	[in] Specifies stride of pixels to be copied to the image.
srcPixels	[in] Points to memory block containing pixels that to be copied to the image.
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is not zero and is less than minimal value for specified pixel format and width. - or - *srcStride* is less than minimal value for specified pixel format and width.
N_E_ARGUMENT_NULL	*srcPixels* or *pHImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is less than zero.

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride function.

Format of memory block *srcPixels* points to must be the same as described in NImageGetPixels function, only stride is equal to *srcStride*.

Created image must be deleted using NImageFree function.

horzResolution and *vertResolution* can be zero if resolution is not applicable for the image.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageCreateWrapper | NImageGetStride | NImageGetPixels

5.3.5.4. NImageCreateFromFile Function

Creates (loads) an image from file of specified format.

```
NResult N_API NImageCreateFromFile(
        const NChar * szFileName,
        HNImageFormat hImageFormat,
        HNImage * pHImage
);
```

Parameters

szFileName	[in] Points to string that specifies file name.
hImageFormat	[in] Handle to the image format of the file. Can be NULL.
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid for specified image format.

Error Code	Condition
N_E_NOT_SUPPORTED	*hImageFormat* is NULL and none of supported image formats is registered with file extension of *szFileName*. - or - *hImageFormat* is NULL and image format registered with file extension of *szFileName* does not support reading. - or - Image format specified by *hImageFormat* does not support reading.

Remarks

If *hImageFormat* is NULL image format is selected by file extension of *szFileName*.

Created image must be deleted using NImageFree function.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageFormatCanRead

5.3.5.5. NImageCreateFromImage Function

Creates an image from specified image with specified pixel format and stride.

```
NResult N_API NImageCreateFromImage(
        NPixelFormat pixelFormat,
        NSizeType stride,
        HNImage hSrcImage,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
stride	[in] Specifies stride of the image. Can be zero.
hSrcImage	[in] Handle to image used as source for the image.

pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is not zero and is less than minimal value for specified pixel format and source image width.
N_E_ARGUMENT_NULL	*hSrcImage* or *pHImage* is NULL.

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride function.

Created image must be deleted using NImageFree function.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageCreateFromImageEx | NImageClone | NImageGetStride

5.3.5.6. NImageCreateFromImageEx Function

Creates an image from specified image with specified pixel format, stride and resolution.

```
NResult N_API NImageCreateFromImageEx(
        NPixelFormat pixelFormat,
        NSizeType stride,
        NFloat horzResolution,
        NFloat vertResolution,
        HNImage hSrcImage,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
stride	[in] Specifies stride of the image. Can be zero.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
hSrcImage	[in] Handle to image used as source for the image.
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is not zero and is less than minimal value for specified pixel format and source image width.
N_E_ARGUMENT_NULL	*hSrcImage* or *pHImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*horzResolution* or *vertResolution* is less than zero.

Remarks

If *stride* is zero then image stride is automatically calculated. For more information on image stride see NImageGetStride function.

Created image must be deleted using NImageFree function.

horzResolution and *vertResolution* can be zero if resolution is not applicable for the image.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageCreateFromImage | NImageClone | NImageGetStride

5.3.5.7. NImageCreateWrapper Function

Creates an image wrapper for specified pixels with specified pixel format, size, stride and resolution.

```
NResult N_API NImageCreateWrapper(
        NPixelFormat pixelFormat,
        NUInt width,
        NUInt height,
        NSizeType stride,
        NFloat horzResolution,
        NFloat vertResolution,
        void * pixels,
        NBool ownsPixels,
        HNImage * pHImage
);
```

Parameters

pixelFormat	[in] Specifies pixel format of the image.
width	[in] Specifies width of the image.
height	[in] Specifies height of the image.
stride	[in] Specifies stride of the image.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
pixels	[in] Points to memory block containing pixels for the image.
ownsPixels	[in] Specifies whether pixels will be automatically deleted with the image (if set to NTrue).
pHImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*pixelFormat* has invalid value. - or - *stride* is less than minimal value for specified pixel format and width.
N_E_ARGUMENT_NULL	*pixels* or *pHImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is less than zero.

Remarks

For more information on image stride see NImageGetStride function.

Format of memory block *pixels* points to must be the same as described in NImageGetPixels function.

Created image must be deleted using NImageFree function.

pixels must not be deleted during lifetime of the image. If *ownsPixels* is NTrue then *pixels* will be automatically deleted with the image.

horzResolution and *vertResolution* can be zero if resolution is not applicable for the image.

See Also

NImage Module | HNImage | NImageFree | NImageCreate | NImageCreateFromData | NImageGetStride | NImageGetPixels

5.3.5.8. NImageFree Function

Deletes the image. After the image is deleted the specified handle is no longer valid.

```
void N_API NImageFree(
```

```
        HNImage hImage
);
```

Parameters

hImage	[in] Handle to the image.

Remarks

If *hImage* is NULL does nothing.

See Also

NImage Module | HNImage | NImageCreate

5.3.5.9. NImageGetHeight Function

Retrieves height of the image.

```
NResult N_API NImageGetHeight(
        HNImage hImage,
        NUInt * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NUInt that receives height of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

See Also

NImage Module | HNImage | NImageGetWidth

5.3.5.10. NImageGetHorzResolution Function

Retrieves horizontal resolution of the image.

```
NResult N_API NImageGetHorzResolution(
        HNImage hImage,
        NFloat * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NFloat that receives horizontal resolution in pixels per inch of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Horizontal resolution equal to zero means that it is not applicable for the image.

See Also

NImage Module | HNImage | `NImageGetVertResolution`

5.3.5.11. NImageGetPixelFormat Function

Retrieves pixel format of the image.

```
NResult N_API NImageGetPixelFormat(
        HNImage hImage,
        NPixelFormat * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NPixelFormat that receives pixel format of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

See Also

NImage Module | HNImage | NPixelFormat

5.3.5.12. NImageGetPixels Function

Retrieves pointer to memory block containing pixels of the image.

```
NResult N_API NImageGetPixels(
        HNImage hImage,
        void * * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to void * that receives pointer to memory block containing pixels of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Memory block containing image pixels is organized as image height rows following each other in top-to-bottom order. Each row occupies image stride bytes and is organized as image width pixels following each other in right-to-left order. Each pixel is described by image pixel format.

For more information see `NImageGetPixelFormat`, `NImageGetWidth`, `NImageGetHeight`, `NImageGetStride`, and `NImageGetSize` functions.

See Also

NImage Module | HNImage | `NImageGetPixelFormat` | `NImageGetWidth` | `NImageGetHeight` | `NImageGetStride` | `NImageGetSize`

5.3.5.13. NImageGetSize Function

Retrieves size of memory block containing pixels of the image.

```
NResult N_API NImageGetSize(
        HNImage hImage,
        NSizeType * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NSizeType that receives size of memory block containing pixels of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Size of memory block containing image pixels is equal to image height multiplied by image stride. For more information see `NImageGetHeight` and `NImageGetStride` functions.

See Also

NImage Module | HNImage | `NImageGetHeight` | `NImageGetStride`

5.3.5.14. NImageGetStride Function

Retrieves stride (size of one row) of the image.

```
NResult N_API NImageGetStride(
        HNImage hImage,
        NSizeType * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NSizeType that receives stride of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Stride (size of one row) of the image depends on image pixel format and width. It can not be less than value obtained with NPixelFormatGetRowSize macro with arguments obtained with `NImageGetPixelFormat` and `NImageGetWidth` functions.

See Also

NImage Module | HNImage | NPixelFormatGetRowSize | `NImageGetPixelFormat` | `NImageGetWidth` | `NImageGetSize`

5.3.5.15. NImageGetVertResolution Function

Retrieves vertical resolution of the image.

```
NResult N_API NImageGetVertResolution(
```

```
        HNImage hImage,
        NFloat * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NFloat that receives vertical resolution in pixels per inch of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

Remarks

Vertical resolution equal to zero means that it is not applicable for the image.

See Also

NImage Module | HNImage | `NImageGetHorzResolution`

5.3.5.16. NImageGetWidth Function

Retrieves width of the image.

```
NResult N_API NImageGetWidth(
        HNImage hImage,
        NUInt * pValue
);
```

Parameters

hImage	[in] Handle to the image.
pValue	[out] Pointer to NUInt that receives width of the image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.

See Also

NImage Module | HNImage | NImageGetHeight | NImageGetStride

5.3.5.17. NImageSaveToFile Function

Saves the image to the file of specified format.

```
NResult N_API NImageSaveToFile(
        HNImage hImage,
        const NChar * szFileName,
        HNImageFormat hImageFormat
);
```

Parameters

hImage	[in] Handle to NImage object.
szFileName	[in] Points to string that specifies file name.
hImageFormat	[in] Handle to the image format of the file. Can be NULL.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *szFileName* is NULL.
N_E_NOT_SUPPORTED	*hImageFormat* is NULL and none of supported image formats is registered with file extension of *szFileName*.

Error Code	Condition
	- or - *hImageFormat* is NULL and image format registered with file extension of *szFileName* does not support writing. - or - Image format specified by *hImageFormat* does not support writing.

Remarks

If *hImageFormat* is NULL image format is selected by file extension of *szFileName*.

See Also

NImage Module | HNImage | NImageCreateFromFile | NImageFormatCanWrite

5.3.6. NImages Module

Provides library registration and other additional functionality.

Header file: NImages.h.

Functions

NImagesGetGrayscaleColorWrapper	Creates color wrapper for grayscale image.

See Also

NImages Library

5.3.6.1. NImagesGetGrayscaleColorWrapper Function

Creates color wrapper for grayscale image.

```
NResult N_API NImagesGetGrayscaleColorWrapper(
        HNImage hImage,
        NRgb minColor,
        NRgb maxColor,
        HNImage * pHDstImage
);
```

Parameters

hImage	[in] Handle to image.
minColor	[in] Specifies color to be used for black color.
maxColor	[in] Specifies color to be used for white color.
pHDstImage	[out] Pointer to HNImage that receives handle to created image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Image specified by *hImage* has non-grayscale pixel format (not npfGrayscale or npfMonochrome).
N_E_ARGUMENT_NULL	*hImage* or *pHDstImage* is NULL.

Remarks

Created image must be deleted using NImageFree function.

Created image is a thin wrapper for specified grayscale image. Therefore *hImage* must not be freed before created image.

Gray values in source image are replaced with according RGB values from range [*minColor*, *maxColor*] in created image.

See Also

NImages Module | HNImage | NImageFree

5.3.7. NMonochromeImage Module

Provides functionality for managing 1-bit monochrome images.

Header file: NMonochromeImage.h.

Functions

NMonochromeImageGetPixel	Retrieves value of pixel at the specified coordinates in 1-bit monochrome image.
NMonochromeImageSetPixel	Sets value of pixel at the specified coordinates in 1-bit monochrome image.

Remarks

This module provides advanced functionality, such as individual pixel value retrieval for image with pixel format equal to npfMonochrome.

See Also

NImages Library | NImage Module

5.3.7.1. NMonochromeImageGetPixel Function

Retrieves value of pixel at the specified coordinates in 1-bit monochrome image.

```
NResult N_API NMonochromeImageGetPixel(
        HNImage hImage,
        NUInt x,
        NUInt y,
        NBool * pValue
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
pValue	[out] Points to NBool that receives pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to npf-Monochrome.

Remarks

If pixel is black then value *pValue* points to receives NFalse and if it is white then value receives NTrue.

See Also

NMonochromeImage Module | HNImage | NMonochromeImageSetPixel

5.3.7.2. NMonochromeImageSetPixel Function

Sets value of pixel at the specified coordinates in 1-bit monochrome image.

```
NResult N_API NMonochromeImageSetPixel(
        HNImage hImage,
        NUInt x,
        NUInt y,
        NBool value
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
value	[in] Specifies new pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to npf-Monochrome.

Remarks

If *value* is NFalse then pixel will be black and if it is NTrue then pixel will be white.

See Also

NMonochromeImage Module | HNImage | NMonochromeImageGetPixel

5.3.8. NPixelFormat Module

Provides functionality for working with image pixel format.

Header file: NPixelFormat.h.

Functions

NPixelFormatGetBitsPerPixel-Func	Used internally in NPixelFormatGetBitsPer-Pixel macro.
NPixelFormatIsValid	Checks if specified pixel format is valid.

Structures

NRgb	Represents an RGB color.

Enumerations

NPixelFormat	Specifies pixel format of each pixel in the image.

Macros

NCalcRowSize	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.
NCalcRowSizeEx	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel and alignment.
NPixelFormatGetBitsPerPixel	Retrieves number of bits used to store a pixel from NPixelFormat.
NPixelFormatGetRowSize	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.
NPixelFormatGetRowSizeEx	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat and alignment.
NRgbConst	Makes NRgb constant with field values provided.

See Also

NImages Library

5.3.8.1. NPixelFormat Enumeration

Specifies pixel format of each pixel in the image.

```
typedef enum NPixelFormat_ { } NPixelFormat;
```

Members

`npfGrayscale`	Each pixel value is stored in 8 bits representing 256 shades of gray.
`npfMonochrome`	Each pixel value is stored in 1 bit representing either black or white color.
`npfRgb`	Each pixel value is stored in 24 bits consisting of three 8-bit values representing red, green and blue color components.

Remarks

Image pixel format is not limited to members of this enumeration. However only these members are provided for usage with this product.

See Also

NPixelFormat Module | HNImage

5.3.8.2. NRgb Structure

Represents an RGB color.

```
typedef struct NRgb_ { } NRgb;
```

Fields

Blue	Blue component value of this NRgb.
Green	Green component value of this NRgb.
Red	Red component value of this NRgb.

See Also

NPixelFormat Module

5.3.8.2.1. NRgb.Blue Field

Blue component value of this NRgb.

```
NByte Blue;
```

See Also

NRgb Structure

5.3.8.2.2. NRgb.Green Field

Green component value of this NRgb.

```
NByte Green;
```

See Also

NRgb Structure

5.3.8.2.3. NRgb.Red Field

Red component value of this NRgb.

```
NByte Red;
```

See Also

NRgb Structure

5.3.9. NRgbImage Module

Provides functionality for managing 24-bit RGB images.

Header file: `NRgbImage.h`.

Functions

`NRgbImageGetPixel`	Retrieves value of pixel at the specified co-ordinates in 24-bit RGB image.
`NRgbImageSetPixel`	Sets value of pixel at the specified coordinates in 24-bit RGB image.

Remarks

This module provides advanced functionality, such as individual pixel value retrieval for image with pixel format equal to `npfRgb`.

See Also

NImages Library | NImage Module

5.3.9.1. NRgbImageGetPixel Function

Retrieves value of pixel at the specified coordinates in 24-bit RGB image.

```
NResult N_API NRgbImageGetPixel(
        HNImage hImage,
        NUInt x,
        NUInt y,
        NRgb * pValue
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.
y	[in] Specifies y-coordinate of the pixel.
pValue	[out] Pointer to NRgb that receives pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to `npfRgb`.

See Also

NRgbImage Module | HNImage | NRgb | `NRgbImageSetPixel`

5.3.9.2. NRgbImageSetPixel Function

Sets value of pixel at the specified coordinates in 24-bit RGB image.

```
NResult N_API NRgbImageSetPixel(
        HNImage hImage,
        NUInt x,
        NUInt y,
        const NRgb * pValue
);
```

Parameters

hImage	[in] Handle to image.
x	[in] Specifies x-coordinate of the pixel.

y	[in] Specifies y-coordinate of the pixel.
pValue	[in] Pointer to NRgb that specifies new pixel value.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hImage* or *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*x* is greater than or equal to image width. - or - *y* is greater than or equal to image height.
N_E_FORMAT	Image pixel format is not equal to `npfRgb`.

See Also

NRgbImage Module | HNImage | NRgb | `NRgbImageGetPixel`

5.3.10. Tiff Module

Provides functionality for loading images in TIFF format.

Header file: `Tiff.h`.

Functions

`TiffLoadImageFromFile`	Loads image from TIFF file.
`TiffLoadImageFromMemory`	Loads image from memory buffer containing TIFF file.

See Also

NImages Library

5.3.10.1. TiffLoadImageFromFile Function

Loads image from TIFF file.

```
NResult N_API TiffLoadImageFromFile(
        const NChar * szFileName,
        HNImage * pHImage
);
```

Parameters

szFileName	[in] Points to string that specifies file name.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*szFileName* or *pHImage* is NULL.
N_E_FORMAT	Format of file specified by *szFileName* is invalid.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Tiff Module | HNImage | `TiffLoadImageFromMemory`

5.3.10.2. TiffLoadImageFromMemory Function

Loads image from memory buffer containing TIFF file.

```
NResult N_API TiffLoadImageFromMemory(
        const void * buffer,
        NSizeType bufferLength,
        HNImage * pHImage
);
```

Parameters

buffer	[in] Pointer to memory buffer.
bufferLength	[in] Length of memory buffer.
pHImage	[out] Pointer to HNImage that receives handle to loaded image.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*buffer* is NULL and *bufferLength* is not equal to zero. - or - *pHImage* is NULL.
N_E_FORMAT	Format of file contained in buffer specified by *buffer* is invalid.

Remarks

This is a low-level function and can be changed in future version of the library.

See Also

Tiff Module | HNImage | `TiffLoadImageFromFile`

5.4. FncExtractor Library

Provides functionality for extracting Neurotechnologija Finger Records from fingerprint images using FingerCell algorithm.

Import library (Windows): `FncExtractor.dll.lib`.

DLL (Windows): `FncExtractor.dll`.

Shared object (Linux): `libFncExtractor.so`.

Requirements (Windows):

- `NCore.dll`.
- `NFRecord.dll`.

Requirements (Linux):

- `libNCore.so`.
- `libNFRecord.so`.

Modules

FncExtractor	Provides functionality for extracting Neurotechnologija Finger Records (NFRecords) from fingerprint images using FingerCell algorithm encapsulated in Neurotechnologija Fingerprint Features Extractor (FncExtractor) object.

5.4.1. FncExtractor Module

Provides functionality for extracting Neurotechnologija Finger Records (NFRecords) from fingerprint images using FingerCell algorithm encapsulated in Neurotechnologija Fingerprint Features Extractor (FncExtractor) object.

Header file: `FncExtractor.h` (includes `FncExtractorParams.h` and `FncExtractorTypes.h`).

Functions

`FnceCopyParameters`	Copies parameter values from one FncExtractor to another.
`FnceCreate`	Creates a FncExtractor.
`FnceExtract`	Extracts a packed NFRecord from the image using the specified FncExtractor.
`FnceExtractFromImage`	Extracts a packed NFRecord from the NImage using the specified FncExtractor.
`FnceExtractUnpacked`	Extracts a NFRecord from the image using the specified FncExtractor.
`FnceExtractUnpackedFromImage`	Extracts a NFRecord from the NImage using

	the specified FncExtractor.
FnceFastExtractEnd	Ends fast extraction.
FnceFastExtractFinish	Finishes fast extraction by extracting a packed NFRecord from the whole image used during start of fast extraction using the specified FncExtractor.
FnceFastExtractFinishUnpacked	Finishes fast extraction by extracting a NFRecord from the whole image used during start of fast extraction using the specified FncExtractor.
FnceFastExtractFinishUnpackedWithImage	Finishes fast extraction by extracting a NFRecord from the whole NImage used during start of fast extraction using the specified FncExtractor.
FnceFastExtractFinishWithImage	Finishes fast extraction by extracting a packed NFRecord from the whole NImage used during start of fast extraction using the specified FncExtractor.
FnceFastExtractStart	Starts fast extraction by extracting a packed NFRecord from a part of the image using the specified FncExtractor.
FnceFastExtractStartFromImage	Starts fast extraction by extracting a packed NFRecord from a part of the NImage using the specified FncExtractor.
FnceFastExtractStartUnpacked	Starts fast extraction by extracting a NFRecord from a part of the image using the specified FncExtractor.
FnceFastExtractStartUnpackedFromImage	Starts fast extraction by extracting a NFRecord from a part of the NImage using the specified FncExtractor.
FnceFree	Deletes the FncExtractor. After the object is deleted the specified handle is no longer valid.
FnceGeneralize	Generalizes count features collections to single features collection.
FnceGeneralizeUnpacked	Generalizes count features collections to single features collection.
FnceGetMaxTemplateSize	Retrieves maximal size of packed NFRecord the specified FncExtractor can extract.

FnceGetParameter	Retrieves value of the specified parameter of the specified FncExtractor.
FnceIsRegistered	Checks if FncExtractor library is registered.
FnceReset	Sets default values for all parameters of the specified FncExtractor.
FnceSetParameter	Sets value of the specified parameter of the specified FncExtractor.

Enumerations

FnceReturnedImage	Specifies kind of image returned after extraction using VFExtractor.
FnceTemplateSize	Specifies a returned NFRecord size from functions: FnceExtract, FnceExtractFromImage, FnceExtractUnpacked, FnceExtractUnpackedFromImage.

Types

HFncExtractor	Handle to FncExtractor object.

Macros

FNCE_MODE_ATMEL_FINGERCHIP	The mode for Atmel FingerChip sensor.
FNCE_MODE_AUTHENTEC_AES2501B	The mode for AuthenTec AES2501B sensor.
FNCE_MODE_AUTHENTEC_AES4000	The mode for AuthenTec AES4000 sensor.
FNCE_MODE_AUTHENTEC_AFS2	The mode for AuthenTec AF-S2 sensor.
FNCE_MODE_BIOMETRIKA_FX2000	The mode for Biometrika FX2000 scanner.
FNCE_MODE_BIOMETRIKA_FX3000	The mode for Biometrika FX3000 scanner.
FNCE_MODE_BMF_BLP100	The mode for BMF BLP100 scanner.
FNCE_MODE_CROSSMATCH_VERIFIER300	The mode for CrossMatch Verifier 300 scanner.

FNCE_MODE_DIGENT_IZZIX	The mode for Digent Izzix FD1000 scanner.
FNCE_MODE_DIGITALPERSONA_UAREU	The mode for Digital Persona U.are.U scanner.
FNCE_MODE_ETHENTICA	The mode for Ethentica scanner.
FNCE_MODE_FUJITSU_MBF200	The mode for Fujitsu MBF200 scanner.
FNCE_MODE_FUTRONIC_FS80	The mode for Futronic's FS80 scanner.
FNCE_MODE_GENERAL	The general mode.
FNCE_MODE_IDENTICATORTECHNOLOGY_DF90	The mode for Identicator Technology DF90 scanner.
FNCE_MODE_IDENTIX_DFR2090	The mode for Identix DFR 2090 scanner.
FNCE_MODE_IDENTIX_TOUCHVIEW	The mode for Identix TouchView scanner.
FNCE_MODE_KEYTRONIC_SECUREDESKTOP	The mode for Keytronic Secure Desktop scanner.
FNCE_MODE_LIGHTUNING_LTTC500	The mode for LighTuning LTT-C500 scanner.
FNCE_MODE_NITGEN_FINGKEY_HAMSTER	The mode for NITGEN Fingkey Hamster scanner.
FNCE_MODE_PRECISEBIOMETRICS_100CS	The mode for Precise Biometrics 100CS scanner.
FNCE_MODE_SECUGEN_HAMSTER	The mode for SecuGen Hamster III scanner.
FNCE_MODE_STARTEK_FM200	The mode for Startek FM200 sensor.
FNCE_MODE_TACOMA_CMOS	The mode for Tacoma CMOS sensor.
FNCE_MODE_TESTECH_BIOI	The mode for Testech Bio-i sensor.
FNCE_MODE_UPEK_TOUCHCHIP	The mode for UPEK TouchChip TCRU1C/TCRU2C sensor.
FNCEP_COPYRIGHT	Identifier specifying library copyright static read-only parameter of type N_TYPE_STRING.
FNCEP_GENERALIZATION_MAXIMAL_ROTATION	Maximal rotation of two features collection to each other. Must be in range 0°..180°.
FNCEP_GENERALIZATION_THRESHOLD	Has the same meaning for features generalization as

	FNCMP_MATCHING_THRESHOLD parameter for features matching.
FNCEP_MODE	Identifier specifying mode (parameter value set) parameter of type N_TYPE_UINT. Parameter value can be one of the FNCE_MODE_XXX.
FNCEP_NAME	Identifier specifying library name static read-only parameter of type N_TYPE_STRING.
FNCEP_QUALITY_THRESHOLD	Identifier specifies image quality threshold.
FNCEP_RETURNED_IMAGE	Identifier specifying kind of image returned after extraction parameter of type N_TYPE_INT. Parameter value can be one of the FnceReturnedImage enumeration members.
FNCEP_TEMPLATE_SIZE	Identifier specifying template size parameter. Parameter value can be one of the FnceTemplateSize enumeration members.
FNCEP_USE_QUALITY	If identifier is set to true(parameter of type N_TYPE_BOOL), then functions `FnceExtract`, `FnceExtractFromImage`, `FnceExtractUnpacked`, `FnceExtractUnpackedFromImage` determines the quality of the image. If the image quality threshold is lower than FNCEP_QUALITY_THRESHOLD the function do not returns template.
FNCEP_VERSION_HIGH	Identifier specifying high part of library version static read-only parameter of type N_TYPE_UINT. Two high-order bytes of parameter value specify major version and two low-order bytes - minor version.
FNCEP_VERSION_LOW	Identifier specifying low part of library version static read-only parameter of type N_TYPE_UINT. Two high-order bytes of parameter value specify major (build) version and two low-order bytes - minor (release) version.

See Also

FncExtractor Library

5.4.1.1. FnceCopyParameters Function

Copies parameter values from one FncExtractor to another.

```
NResult N_API FnceCopyParameters(
        HFncExtractor hDstExtractor,
        HFncExtractor hSrcExtractor
);
```

Parameters

hDstExtractor	[in] Handle to the destination FncExtractor object.
hSrcExtractor	[in] Handle to the source FncExtractor object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hDstExtractor* or *hSrcExtractor* is NULL.

See Also

FncExtractor Module | HFncExtractor

5.4.1.2. FnceCreate Function

Creates a FncExtractor.

```
NResult N_API FnceCreate(
        HFncExtractor * pHExtractor
);
```

Parameters

pHExtractor	[out] Pointer to HFncExtractor that receives

	handle to created FncExtractor object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pHExtractor* is NULL.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using `FnceFree` function.

See Also

FncExtractor Module | HFncExtractor | `FnceFree`

5.4.1.3. FnceExtract Function

Extracts a packed NFRecord from the image using the specified FncExtractor.

```
NResult N_API FnceExtract(
        HFncExtractor hExtractor,
        NUShort width,
        NUShort height,
        NSizeType stride,
        NUShort horzResolution,
        NUShort vertResolution,
        NByte * pixels,
        NFPosition position,
        NFImpressionType impressionType,
        void * buffer,
        NSizeType bufferSize,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
width	[in] Specifies image width.

height	[in] Specifies image height.
stride	[in] Specifies length of the image row in bytes.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
pixels	[in, out] Pointer to memory block containing image pixels.
position	[in] Specifies finger position.
impressionType	[in] Specifies impression type.
buffer	[out] Pointer to memory buffer to store extracted packed NFRecord.
bufferSize	[in] Size of memory buffer to store extracted packed NFRecord.
pSize	[out] Pointer to NSizeType that receives size of extracted packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*stride* is less than *width*. - or - *position* is invalid. - or - *impressionType* is invalid. - or - *bufferSize* is less than needed to store extracted packed NFRecord.

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor*, *pixels*, *buffer* or *pSize* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is out of supported range.
N_E_INVALID_OPERATION	Fast extraction is started.

Remarks

position and *impressionType* are written to extracted NFRecord.

Memory for *buffer* must be allocated before calling the function. FnceGetMaxTemplateSize function should be used to learn the size needed for allocation.

If image is of low quality then *buffer* is not used and value received through *pSize* is zero.

See Also

FncExtractor Module | HFncExtractor | FnceGetMaxTemplateSize | FnceExtractUnpacked | FnceExtractFromImage

5.4.1.4. FnceExtractFromImage Function

Extracts a packed NFRecord from the NImage using the specified FncExtractor.

```
NResult N_API FnceExtractFromImage(
        HFncExtractor hExtractor,
        HNImage hImage,
        NFPosition position,
        NFImpressionType impressionType,
        void * buffer,
        NSizeType bufferSize,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
hImage	[in] Handle to the image.

position	[in] Specifies finger position.
impressionType	[in] Specifies impression type.
buffer	[out] Pointer to memory buffer to store extracted packed NFRecord.
bufferSize	[in] Size of memory buffer to store extracted packed NFRecord.
pSize	[out] Pointer to NSizeType that receives size of extracted packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*hImage* is invalid. - or - *position* is invalid. - or - *impressionType* is invalid. - or - *bufferSize* is less than needed to store extracted packed NFRecord.
N_E_ARGUMENT_NULL	*hExtractor*, *hImage*, *buffer* or *pSize* is NULL.
N_E_INVALID_OPERATION	Fast extraction is started.

Remarks

position and *impressionType* are written to extracted NFRecord.

Memory for *buffer* must be allocated before calling the function. FnceGetMaxTemplateSize function should be used to learn the size needed for allocation.

If image is of low quality then *buffer* is not used and value received through *pSize* is zero.

See Also

FncExtractor Module | HFncExtractor | HNImage | FnceGetMaxTemplateSize | FnceExtract | FnceExtractUnpackedFromImage

5.4.1.5. FnceExtractUnpacked Function

Extracts a NFRecord from the image using the specified FncExtractor.

```
NResult N_API FnceExtractUnpacked(
        HFncExtractor hExtractor,
        NUShort width,
        NUShort height,
        NSizeType stride,
        NUShort horzResolution,
        NUShort vertResolution,
        NByte * pixels,
        NFPosition position,
        NFImpressionType impressionType,
        HNFRecord * pHRecord
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
width	[in] Specifies image width.
height	[in] Specifies image height.
stride	[in] Specifies length of the image row in bytes.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
pixels	[in, out] Pointer to memory block containing image pixels.
position	[in] Specifies finger position.
impressionType	[in] Specifies impression type.
pHRecord	[out] Pointer to HNFRecord that receives handle to created NFRecord object contain-

	ing extracted template.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*stride* is less than *width*. - or - *position* is invalid. - or - *impressionType* is invalid.
N_E_ARGUMENT_NULL	*hExtractor*, *pixels* or *pHRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is out of supported range.
N_E_INVALID_OPERATION	Fast extraction is started.

Remarks

position and *impressionType* are written to extracted NFRecord.

If image is of low quality then handle received through *pHRecord* is NULL.

See Also

FncExtractor Module | HFncExtractor | HNFRecord | FnceExtract | FnceExtractUnpackedFromImage

5.4.1.6. FnceExtractUnpackedFromImage Function

Extracts a NFRecord from the NImage using the specified FncExtractor.


```
NResult N_API FnceExtractUnpackedFromImage(
        HFncExtractor hExtractor,
        HNImage hImage,
        NFPosition position,
        NFImpressionType impressionType,
        HNFRecord * pHRecord
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
hImage	[in] Handle to the image.
position	[in] Specifies finger position.
impressionType	[in] Specifies impression type.
pHRecord	[out] Pointer to HNFRecord that receives handle to created NFRecord object containing extracted template.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*hImage* is invalid. - or - *position* is invalid. - or - *impressionType* is invalid.
N_E_ARGUMENT_NULL	*hExtractor*, *hImage* or *pHRecord* is NULL.
N_E_INVALID_OPERATION	Fast extraction is started.

Remarks

position and *impressionType* are written to extracted NFRecord.

If image is of low quality then handle received through *pHRecord* is NULL.

See Also

FncExtractor Module | HFncExtractor | HNImage | HNFRecord | FnceExtractFromImage | FnceExtractUnpacked

5.4.1.7. FnceFastExtractEnd Function

Ends fast extraction.

```
NResult N_API FnceFastExtractEnd(
        HFncExtractor hExtractor
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor*, *hImage* or *pHRecord* is NULL.
N_E_INVALID_OPERATION	Fast extraction is not started.

See Also

FncExtractor Module | HFncExtractor | FnceFastExtractStartFromImage | FnceFastExtractStartUnpackedFromImage | FnceFastExtractFinishWithImage | FnceFastExtractFinishUnpackedWithImage

5.4.1.8. FnceFastExtractFinish Function

Finishes fast extraction by extracting a packed NFRecord from the whole image used during start of fast extraction using the specified FncExtractor.

```
NResult N_API FnceFastExtractFinish(
        HFncExtractor hExtractor,
        NUShort width,
```



```
        NUShort height,
        NSizeType stride,
        NUShort horzResolution,
        NUShort vertResolution,
        NByte * pixels,
        void * buffer,
        NSizeType bufferSize,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
width	[in] Specifies image width.
height	[in] Specifies image height.
stride	[in] Specifies length of the image row in bytes.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
pixels	[in, out] Pointer to memory block containing image pixels. Can be NULL.
buffer	[out] Pointer to memory buffer to store extracted packed NFRecord.
bufferSize	[in] Size of memory buffer to store extracted packed NFRecord.
pSize	[out] Pointer to NSizeType that receives size of extracted packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	**Condition**
N_E_ARGUMENT	*stride* is less than *width*. - or -

Error Code	Condition
	bufferSize is less than needed to store extracted packed NFRecord.
N_E_ARGUMENT_NULL	*hExtractor*, *pixels*, *buffer* or *pSize* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is out of supported range.
N_E_INVALID_OPERATION	Fast extraction is started.

Remarks

If *pixels* is NULL then *width*, *height*, *stride*, *horzResolution* and *vertResolution* are ignored.

Memory for *buffer* must be allocated before calling the function. FnceGetMaxTemplateSize function should be used to learn the size needed for allocation.

If image is of low quality then *buffer* is not used and value received through *pSize* is zero.

See Also

FncExtractor Module | HFncExtractor | FnceGetMaxTemplateSize | FnceFastExtractFinishUnpacked | FnceFastExtractFinishWithImage | FnceFastExtractStart | FnceFastExtractEnd

5.4.1.9. FnceFastExtractFinishUnpacked Function

Finishes fast extraction by extracting a NFRecord from the whole image used during start of fast extraction using the specified FncExtractor.

```
NResult N_API FnceFastExtractFinishUnpacked(
        HFncExtractor hExtractor,
        NUShort width,
        NUShort height,
        NSizeType stride,
        NUShort horzResolution,
        NUShort vertResolution,
        NByte * pixels,
        HNFRecord * pHRecord
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
width	[in] Specifies image width.
height	[in] Specifies image height.
stride	[in] Specifies length of the image row in bytes.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
pixels	[in, out] Pointer to memory block containing image pixels. Can be NULL.
pHRecord	[out] Pointer to HNFRecord that receives handle to created NFRecord object containing extracted template.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*stride* is less than *width*.
N_E_ARGUMENT_NULL	*hExtractor*, *pixels* or *pHRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is out of supported range.
N_E_INVALID_OPERATION	Fast extraction is started.

Remarks

If *pixels* is NULL then *width*, *height*, *stride*, *horzResolution* and *vertResolution* are ignored.

If image is of low quality then handle received through *pHRecord* is NULL.

See Also

FncExtractor Module | HFncExtractor | HNFRecord | FnceFastExtractFinish | FnceFastExtractFinishUnpackedWithImage | FnceFastExtractStartUnpacked | FnceFastExtractEnd

5.4.1.10. FnceFastExtractFinishUnpackedWithImage Function

Finishes fast extraction by extracting a NFRecord from the whole NImage used during start of fast extraction using the specified FncExtractor.

```
NResult N_API FnceFastExtractFinishUnpackedWithImage(
        HFncExtractor hExtractor,
        HNImage hImage,
        HNFRecord * pHRecord
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
hImage	[in] Handle to the image used during start of fast extraction. Can be NULL.
pHRecord	[out] Pointer to HNFRecord that receives handle to created NFRecord object containing extracted template.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*hImage* is invalid.
N_E_ARGUMENT_NULL	*hExtractor* or *pHRecord* is NULL.
N_E_INVALID_OPERATION	Fast extraction is not started.

Remarks

If image is of low quality then handle received through *pHRecord* is NULL.

See Also

FncExtractor Module | HFncExtractor | HNImage | HNFRecord | FnceFastExtractFinishWithImage | FnceFastExtractFinishUnpacked | FnceFastExtractStartUnpackedFromImage | FnceFastExtractEnd

5.4.1.11. FnceFastExtractFinishWithImage Function

Finishes fast extraction by extracting a packed NFRecord from the whole NImage used during start of fast extraction using the specified FncExtractor.

```
NResult N_API FnceFastExtractFinishWithImage(
        HFncExtractor hExtractor,
        HNImage hImage,
        void * buffer,
        NSizeType bufferSize,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
hImage	[in] Handle to the image used during start of fast extraction. Can be NULL.
buffer	[out] Pointer to memory buffer to store extracted packed NFRecord.
bufferSize	[in] Size of memory buffer to store extracted packed NFRecord.
pSize	[out] Pointer to NSizeType that receives size of extracted packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*hImage* is invalid.

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor*, *buffer* or *pSize* is NULL.
N_E_INVALID_OPERATION	Fast extraction is not started.

Remarks

Memory for *buffer* must be allocated before calling the function. FnceGetMaxTemplateSize function should be used to learn the size needed for allocation.

If image is of low quality then handle received through *pHRecord* is NULL.

See Also

FncExtractor Module | HFncExtractor | HNImage | FnceGetMaxTemplateSize | FnceFastExtractFinishUnpackedWithImage | FnceFastExtractFinish | FnceFastExtractStartFromImage | FnceFastExtractEnd

5.4.1.12. FnceFastExtractStart Function

Starts fast extraction by extracting a packed NFRecord from a part of the image using the specified FncExtractor.

```
NResult N_API FnceFastExtractStart(
        HFncExtractor hExtractor,
        NUShort width,
        NUShort height,
        NSizeType stride,
        NUShort horzResolution,
        NUShort vertResolution,
        NByte * pixels,
        NFPosition position,
        NFImpressionType impressionType,
        void * buffer,
        NSizeType bufferSize,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
width	[in] Specifies image width.
height	[in] Specifies image height.
stride	[in] Specifies length of the image row in

	bytes.
horzResolution	[in] Specifies horizontal resolution in pixels per inch of the image.
vertResolution	[in] Specifies vertical resolution in pixels per inch of the image.
pixels	[in, out] Pointer to memory block containing image pixels.
position	[in] Specifies finger position.
impressionType	[in] Specifies impression type.
buffer	[out] Pointer to memory buffer to store extracted packed NFRecord.
bufferSize	[in] Size of memory buffer to store extracted packed NFRecord.
pSize	[out] Pointer to NSizeType that receives size of extracted packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	**Condition**
N_E_ARGUMENT	*stride* is less than *width*. - or - *position* is invalid. - or - *impressionType* is invalid. - or - *bufferSize* is less than needed to store extracted packed NFRecord.
N_E_ARGUMENT_NULL	*hExtractor*, *pixels*, *buffer* or *pSize* is NULL.

Error Code	Condition
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is out of supported range.
N_E_INVALID_OPERATION	Fast extraction is started.

Remarks

position and *impressionType* are written to extracted NFRecord.

Memory for *buffer* must be allocated before calling the function. FnceGetMaxTemplateSize function should be used to learn the size needed for allocation.

If image is of low quality then *buffer* is not used and value received through *pSize* is zero.

Fast extraction must be either end with FnceFastExtractEnd function or finished with for example FnceFastExtractFinish function.

See Also

FncExtractor Module | HFncExtractor | FnceGetMaxTemplateSize | FnceFastExtractStartUnpacked | FnceFastExtractStartFromImage | FnceFastExtractFinish | FnceFastExtractEnd

5.4.1.13. FnceFastExtractStartFromImage Function

Starts fast extraction by extracting a packed NFRecord from a part of the NImage using the specified FncExtractor.

```
NResult N_API FnceFastExtractStartFromImage(
        HFncExtractor hExtractor,
        HNImage hImage,
        NFPosition position,
        NFImpressionType impressionType,
        void * buffer,
        NSizeType bufferSize,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.

hImage	[in] Handle to the image.
position	[in] Specifies finger position.
impressionType	[in] Specifies impression type.
buffer	[out] Pointer to memory buffer to store extracted packed NFRecord.
bufferSize	[in] Size of memory buffer to store extracted packed NFRecord.
pSize	[out] Pointer to NSizeType that receives size of extracted packed NFRecord.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*hImage* is invalid. - or - *position* is invalid. - or - *impressionType* is invalid. - or - *bufferSize* is less than needed to store extracted packed NFRecord.
N_E_ARGUMENT_NULL	*hExtractor*, *hImage*, *buffer* or *pSize* is NULL.
N_E_INVALID_OPERATION	Fast extraction is already started.

Remarks

position and *impressionType* are written to extracted NFRecord.

Memory for *buffer* must be allocated before calling the function. FnceGetMaxTem-

`plateSize` function should be used to learn the size needed for allocation.

If image is of low quality then handle received through *`pHRecord`* is NULL.

Fast extraction must be either end with `FnceFastExtractEnd` function or finished with for example `FnceFastExtractFinishWithImage` function.

See Also

FncExtractor Module | HFncExtractor | HNImage | `FnceGetMaxTemplateSize` | `FnceFastExtractStartUnpackedFromImage` | `FnceFastExtractStart` | `FnceFastExtractFinishWithImage` | `FnceFastExtractEnd`

5.4.1.14. FnceFastExtractStartUnpacked Function

Starts fast extraction by extracting a NFRecord from a part of the image using the specified FncExtractor.

```
NResult N_API FnceFastExtractStartUnpacked(
        HFncExtractor hExtractor,
        NUShort width,
        NUShort height,
        NSizeType stride,
        NUShort horzResolution,
        NUShort vertResolution,
        NByte * pixels,
        NFPosition position,
        NFImpressionType impressionType,
        HNFRecord * pHRecord
);
```

Parameters

`hExtractor`	[in] Handle to the FncExtractor object.
`width`	[in] Specifies image width.
`height`	[in] Specifies image height.
`stride`	[in] Specifies length of the image row in bytes.
`horzResolution`	[in] Specifies horizontal resolution in pixels per inch of the image.
`vertResolution`	[in] Specifies vertical resolution in pixels per inch of the image.
`pixels`	[in, out] Pointer to memory block containing image pixels.
`position`	[in] Specifies finger position.

impressionType	[in] Specifies impression type.
pHRecord	[out] Pointer to HNFRecord that receives handle to created NFRecord object containing extracted template.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*stride* is less than *width*. - or - *position* is invalid. - or - *impressionType* is invalid.
N_E_ARGUMENT_NULL	*hExtractor*, *pixels* or *pHRecord* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*width* or *height* is zero. - or - *horzResolution* or *vertResolution* is out of supported range.
N_E_INVALID_OPERATION	Fast extraction is started.

Remarks

position and *impressionType* are written to extracted NFRecord.

If image is of low quality then handle received through *pHRecord* is NULL.

Fast extraction must be either end with FnceFastExtractEnd function or finished with for example FnceFastExtractFinishUnpacked function.

See Also

FncExtractor Module | HFncExtractor | HNFRecord | FnceFastExtractStart | FnceFastExtractStartUnpackedFromImage | FnceFastExtractFinishUnpacked | FnceFastExtractEnd

5.4.1.15. FnceFastExtractStartUnpackedFromImage Function

Starts fast extraction by extracting a NFRecord from a part of the NImage using the specified FncExtractor.

```
NResult N_API FnceFastExtractStartUnpackedFromImage(
        HFncExtractor hExtractor,
        HNImage hImage,
        NFPosition position,
        NFImpressionType impressionType,
        HNFRecord * pHRecord
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
hImage	[in] Handle to the image.
position	[in] Specifies finger position.
impressionType	[in] Specifies impression type.
pHRecord	[out] Pointer to HNFRecord that receives handle to created NFRecord object containing extracted template.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	*hImage* is invalid. - or - *position* is invalid. - or - *impressionType* is invalid.

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor*, *hImage* or *pHRecord* is NULL.
N_E_INVALID_OPERATION	Fast extraction is already started.

Remarks

position and *impressionType* are written to extracted NFRecord.

If image is of low quality then handle received through *pHRecord* is NULL.

Fast extraction must be either end with FnceFastExtractEnd function or finished with for example FnceFastExtractFinishUnpackedWithImage function.

See Also

FncExtractor Module | HFncExtractor | HNImage | HNFRecord | FnceFastExtract-StartFromImage | FnceFastExtractStartUnpacked | FnceFastExtract-FinishUnpackedWithImage | FnceFastExtractEnd

5.4.1.16. FnceFree Function

Deletes the FncExtractor. After the object is deleted the specified handle is no longer valid.

```
void N_API FnceFree(
        HFncExtractor hExtractor
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.

Remarks

If *hExtractor* is NULL, does nothing.

See Also

FncExtractor Module | HFncExtractor

5.4.1.17. FnceGeneralize Function

Generalizes count features collections to single features collection.

```
NResult N_API FncGeneralize(
        HFncExtractor hExtractor,
        NInt templateCount,
        const void * * arTemplates,
        const NSizeType * arTemplateSizes,
        void * buffer,
        NSizeType bufferSize,
        NSizeType * pSize,
        NInt * pBaseTemplateIndex
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
templateCount	[in] The templates count.
arTemplates	[in] Pointer to void * that receives pointer to memory block containing templates array.
arTemplateSizes	[in] Pointer to array of NSizeType that contains sizes of each template.
buffer	[out] Pointer to memory buffer to store generalized template.
bufferSize	[in] Size of memory buffer to store generalized template.
pSize	[out] Pointer to NSizeType that receives size of template.
pBaseTemplateIndex	Index of main generalization template.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	**Condition**
N_E_ARGUMENT	One of *arTemplates* template is NULL.
N_E_ARGUMENT_NULL	*hExtractor*, *arTemplates*, *arTemplateSizes*, *buffer*, *pSize*, or *pBaseTemplateIndex* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*templateCount* is less or greater than NFG_MIN_TEMPLATES.

Error Code	Condition
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

If templates can not be generalized then *buffer* is not used, value received through *pSize* is zero and value received through *pBaseTemplateIndex* is -1.

See Also

FncExtractor Module | HFncExtractor | HNFRecord | FnceExtract | FnceGeneralizeUnpacked

5.4.1.18. FnceGeneralizeUnpacked Function

Generalizes count features collections to single features collection.

```
NResult N_API FnceGeneralizeUnpacked(
        HFncExtractor hExtractor,
        NInt templateCount,
        const void * * arTemplates,
        const NSizeType * arTemplateSizes,
        HNFRecord * pHRecord,
        NInt * pBaseTemplateIndex
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
templateCount	[in] The templates count.
arTemplates	[in] Pointer to void * that receives pointer to memory block containing templates array.
arTemplateSizes	[in] Pointer to array of NSizeType that contains sizes of each template.
pHRecord	[out] Pointer to HNFRecord that receives handle to created NFRecord object containing generalized template.
pBaseTemplateIndex	Index of main generalization template.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	One of *arTemplates* template is NULL.
N_E_ARGUMENT_NULL	*hExtractor*, *arTemplates*, *arTemplateSizes*, *pHRecord*, or *pBaseTemplateIndex* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	*templateCount* is less or greater than NFG_MIN_TEMPLATES.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

If templates can not be generalized then value received through *pHRecord* is NULL and value received through *pBaseTemplateIndex* is -1.

See Also

FncExtractor Module | HFncExtractor | HNFRecord | FnceExtract | FnceGeneralize

5.4.1.19. FnceGetMaxTemplateSize Function

Retrieves maximal size of packed NFRecord the specified FncExtractor can extract.

```
NResult N_API FnceGetMaxTemplateSize(
        HFncExtractor hExtractor,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.
pSize	[out] Pointer to NSizeType that receives maximal template size.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* or *pSize* is NULL.

See Also

FncExtractor Module | HFncExtractor

5.4.1.20. FnceGetParameter Function

Retrieves value of the specified parameter of the specified FncExtractor.

```
NResult N_API FnceGetParameter(
        HFncExtractor hExtractor,
        NUInt parameterId,
        void * pValue
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object. Can be NULL if retrieving static parameter value.
parameterId	[in] Identifier of the parameter to retrieve.
pValue	[out] Pointer to variable that receives parameter value.

Return Values

If the function succeeds and *parameterId* specifies a N_TYPE_STRING type parameter, and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*parameterId* specifies a non-static parameter and *hExtractor* is NULL. - or - *parameterId* specifies a

Error Code	Condition
	N_TYPE_STRING type parameter and *pValue* is NULL.
N_E_PARAMETER	*parameterId* is invalid.

Remarks

The following values can be used for *parameterId*:

- FNCEP_COPYRIGHT
- FNCEP_GENERALIZATION_THRESHOLD
- FNCEP_GENERALIZATION_MAXIMAL_ROTATION
- FNCEP_MODE
- FNCEP_NAME
- FNCEP_USE_QUALITY
- FNCEP_QUALITY_THRESHOLD
- FNCEP_RETURNED_IMAGE
- FNCEP_TEMPLATE_SIZE
- FNCEP_VERSION_HIGH
- FNCEP_VERSION_LOW

To learn the type of the parameter pass value obtained with NParameterMakeId macro using N_PC_TYPE_ID code and the parameter id via *parameterId* parameter and pointer to NInt that will receive one of N_TYPE_XXX via *pValue* parameter. *hExtractor* can be NULL in this case.

See Also

FncExtractor Module | HFncExtractor | FnceSetParameter

5.4.1.21. FnceIsRegistered Function

Checks if FncExtractor library is registered.

```
NBool N_API FnceIsRegistered(void);
```

Return Values

NTrue if library is registered, NFalse otherwise.

See Also

FncExtractor Module

5.4.1.22. FnceReset Function

Sets default values for all parameters of the specified FncExtractor.

```
NResult N_API FnceReset(
        HFncExtractor hExtractor
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hExtractor* is NULL.

See Also

FncExtractor Module | HFncExtractor

5.4.1.23. FnceReturnedImage Enumeration

Specifies kind of image returned after extraction using VFExtractor.

```
typedef enum FnceReturnedImage_ { } FnceReturnedImage;
```

Members

vferiBinarized	Binarized (filtered) image is written to the image used for extraction.
vferiNone	The image used for extraction is left unchanged.
vferiSkeletonized	Skeletonized image is written to the image used for extraction.

See Also

FncExtractor Module

5.4.1.24. FnceSetParameter Function

Sets value of the specified parameter of the specified FncExtractor.

```
NResult N_API FnceSetParameter(
        HFncExtractor hExtractor,
        NUInt parameterId,
        const void * pValue
);
```

Parameters

hExtractor	[in] Handle to the FncExtractor object. Can be NULL if setting static parameter value.
parameterId	[in] Identifier of the parameter to set.
pValue	[in] Pointer to the parameter value to set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	Argument value is out of range.
N_E_INVALID_OPERATION	Fast extraction is started.
N_E_PARAMETER	Parameter ID is invalid.

Remarks

The following values can be used for *parameterId*:

- FNCEP_GENERALIZATION_THRESHOLD
- FNCEP_GENERALIZATION_MAXIMAL_ROTATION
- FNCEP_MODE
- FNCEP_USE_QUALITY

- FNCEP_QUALITY_THRESHOLD
- FNCEP_RETURNED_IMAGE
- FNCEP_TEMPLATE_SIZE

To learn the type of the parameter pass value obtained with NParameterMakeId macro using N_PC_TYPE_ID code and the parameter id via *parameterId* parameter and pointer to NInt that will receive one of N_TYPE_XXX via *pValue* parameter. *hExtractor* can be NULL in this case.

See Also

FncExtractor Module | HFncExtractor | FnceGetParameter

5.4.1.25. FnceTemplateSize Enumeration

Specifies a returned NFRecord size from functions: FnceExtract, FnceExtractFromImage, FnceExtractUnpacked, FnceExtractUnpackedFromImage.

```
typedef enum FnceTemplateSize_ { } FnceTemplateSize;
```

Members

vfetsLarge	The large NFRecord size.
vfetsSmall	The small NFRecord size.

See Also

FncExtractor Module

5.5. FncMatcher Library

Provides functionality for comparing Neurotechnologija Finger Records using FaceCell algorithm.

Import library (Windows): FncMatcher.dll.lib.

DLL (Windows): FncMatcher.dll.

Shared object (Linux): libFncMatcher.so.

Requirements (Windows):

- NCore.dll.

- `NFRecord.dll`.

Requirements (Linux):

- `libNCore.so`.
- `libNFRecord.so`.

Modules

FncMatcher	Provides functionality for comparing Neurotechnologija Finger Records (NFRecords) using FingerCell algorithm encapsulated in Neurotechnologija Finger Matcher Fnc (FncMatcher) object.

5.5.1. FncMatcher Module

Provides functionality for comparing Neurotechnologija Finger Records (NFRecords) using FingerCell algorithm encapsulated in Neurotechnologija Finger Matcher Fnc (FncMatcher) object.

Header file: `FncMatcher.h` (includes `FncMatcherParams.h` and `FncmMatchDetails.h`).

Functions

`FncmCopyParameters`	Copies parameter values from one FncMatcher to another.
`FncmCreate`	Creates a FncMatcher.
`FncmMatchDetailsDeserialize`	Deserializes data from buffer into FncmMatchDetails structure.
`FncmFree`	Deletes the FncMatcher. After the object is deleted the specified handle is no longer valid.
`FncmGetParameter`	Retrieves value of the specified parameter of the specified FncMatcher.
`FncmIdentifyEnd`	Ends identification using the specified FncMatcher.
`FncmIdentifyNext`	Compares the specified packed NFRecord with the one identification was started with

	using the specified FncMatcher.
FncmIdentifyStart	Starts identification with the specified packed NFRecord using the specified FncMatcher.
FncmIsRegistered	Checks if FncMatcher library is registered.
FncmMatchDetailsFree	Deletes FncmMatchDetails.
FncmReset	Sets default values for all parameters of the specified FncMatcher.
FncmMatchDetailsSerialize	Serializes FncmMatchDetails structure into buffer.
FncmSetParameter	Sets value of the specified parameter of the specified FncMatcher.
FncmVerify	Compares two packed NFRecords using the specified FncMatcher.

Structures

FncmMatchDetails	Represents details of matching performed with FncMatcher.

Types

HFncMatcher	Handle to FncMatcher object.

See Also

FncMatcher Library

Macros

FNCM_MODE_ATMEL_FINGERCHIP	The mode for Atmel FingerChip sensor.
FNCM_MODE_AUTHENTEC_AES2501B	The mode for AuthenTec AES2501B sensor.
FNCM_MODE_AUTHENTEC_AES4000	The mode for AuthenTec AES4000 sensor.
FNCM_MODE_AUTHENTEC_AFS2	The mode for AuthenTec AF-S2 sensor.

FNCM_MODE_BIOMETRIKA_FX2000	The mode for Biometrika FX2000 scanner.
FNCM_MODE_BIOMETRIKA_FX3000	The mode for Biometrika FX3000 scanner.
FNCM_MODE_BMF_BLP100	The mode for BMF BLP100 scanner.
FNCM_MODE_CROSSMATCH_VERIFIER300	The mode for CrossMatch Verifier 300 scanner.
FNCM_MODE_DIGENT_IZZIX	The mode for Digent Izzix FD1000 scanner.
FNCM_MODE_DIGITALPERSONA_UAREU	The mode for Digital Persona U.are.U scanner.
FNCM_MODE_ETHENTICA	The mode for Ethentica scanner.
FNCM_MODE_FUJITSU_MBF200	The mode for Fujitsu MBF200 scanner.
FNCM_MODE_FUTRONIC_FS80	The mode for Futronic's FS80 scanner.
FNCM_MODE_GENERAL	The general mode.
FNCM_MODE_IDENTICATORTECHNOLOGY_DF90	The mode for Identicator Technology DF90 scanner.
FNCM_MODE_IDENTIX_DFR2090	The mode for Identix DFR 2090 scanner.
FNCM_MODE_IDENTIX_TOUCHVIEW	The mode for Identix TouchView scanner.
FNCM_MODE_KEYTRONIC_SECUREDESKTOP	The mode for Keytronic Secure Desktop scanner.
FNCM_MODE_LIGHTUNING_LTTC500	The mode for LighTuning LTT-C500 scanner.
FNCM_MODE_NITGEN_FINGKEY_HAMSTER	The mode for NITGEN Fingkey Hamster scanner.
FNCM_MODE_PRECISEBIOMETRICS_100CS	The mode for Precise Biometrics 100CS scanner.
FNCM_MODE_SECUGEN_HAMSTER	The mode for SecuGen Hamster III scanner.
FNCM_MODE_STARTEK_FM200	The mode for Startek FM200 sensor.
FNCM_MODE_TACOMA_CMOS	The mode for Tacoma CMOS sensor.
FNCM_MODE_TESTECH_BIOI	The mode for Testech Bio-i sensor.
FNCM_MODE_UPEK_TOUCHCHIP	The mode for UPEK TouchChip TCRU1C/TCRU2C sensors.
FNCMP_COPYRIGHT	Identifier specifying library copyright static read-only parameter of type

	N_TYPE_STRING.
FNCMP_MATCHING_THRESHOLD	Identifier specifying matching threshold (biggest allowed FAR) parameter of type N_TYPE_INT. Parameter value is equal to -12 * log10(FAR) and must be not less than zero (for example, 48 for FAR = 0.01%).
FNCMP_MAXIMAL_ROTATION	Identifier specifying modulus of maximal rotation allowed between two matched NFRecords parameter of type N_TYPE_BYTE. Parameter value is specified in 180/128 degrees units and can not be greater than 128 (+-180 degrees).
FNCMP_MODE	Identifier specifying mode (parameter value set) parameter of type N_TYPE_UINT. Parameter value can be one of the FNCM_MODE_XXX.
FNCMP_NAME	Identifier specifying library name static read-only parameter of type N_TYPE_STRING.
FNCMP_VERSION_HIGH	Identifier specifying high part of library version static read-only parameter of type N_TYPE_UINT. Two high-order bytes of parameter value specify major version and two low-order bytes - minor version.
FNCMP_VERSION_LOW	Identifier specifying low part of library version static read-only parameter of type N_TYPE_UINT. Two high-order bytes of parameter value specify major (build) version and two low-order bytes - minor (release) version.

5.5.1.1. FncmCopyParameters Function

Copies parameter values from one FncMatcher to another.

```
NResult N_API FncmCopyParameters(
        HFncMatcher hDstMatcher,
        HFncMatcher hSrcMatcher
);
```

Parameters

hDstMatcher	[in] Handle to the destination FncMatcher object.
hSrcMatcher	[in] Handle to the source FncMatcher object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hDstMatcher* or *hSrcMatcher* is NULL.
N_E_INVALID_OPERATION	Identification is started on *hDstMatcher*.

See Also

FncMatcher Module | HFncMatcher

5.5.1.2. FncmCreate Function

Creates a FncMatcher.

```
NResult N_API FncmCreate(
        HFncMatcher * pHMatcher
);
```

Parameters

pHMatcher	[out] Pointer to HFncMatcher that receives handle to created FncMatcher object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pHMatcher* is NULL.

Error Code	Condition
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using `FncmFree` function.

See Also

FncMatcher Module | HFncMatcher | `FncmFree`

5.5.1.3. FncmMatchDetailsDeserialize Function

Deserializes data from buffer into FncmMatchDetails structure.

```
NResult N_API FncmMatchDetailsDeerialize(
        conts void * buffer,
        NSizeType BufferLength,
        FncmMatchDetails * * ppMatchDetails
);
```

Parameters

buffer	[in] Pointer to buffer containint serialized matching details
bufferLength	[in] length of the buffer containing the serialized matching details.
ppMatchDetails	[out] Pointer to FncmMatchDetails structure where the deserialized match details are put.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pMatchDetails* or *pBuffer* is NULL.
N_E_FORMA	Data in *buffer* is inconsistent with FncmMatchDetails structure format.

Remarks

Memory for ppMatchDetails is allocated automatically so after this structure is not necessary it must be freed using `FncmMatchDetailsFree` function.

See Also

FncMatcher Module | HFncMatcher | FncmMatchDetails | `FncmMatchDetailsFree`

5.5.1.4. FncmFree Function

Deletes the FncMatcher. After the object is deleted the specified handle is no longer valid.

```
void N_API FncmFree(
        HFncMatcher hMatcher
);
```

Parameters

hMatcher	[in] Handle to FncMatcher object.

Remarks

If *hMatcher* is NULL, does nothing.

See Also

FncMatcher Module | HFncMatcher

5.5.1.5. FncmGetParameter Function

Retrieves value of the specified parameter of the specified FncMatcher.

```
NResult N_API FncmGetParameter(
        HFncMatcher hMatcher,
        NUInt parameterId,
        void * pValue
);
```

Parameters

hMatcher	[in] Handle to the FncMatcher object. Can be NULL if retrieving static parameter value.
parameterId	[in] Identifier of the parameter to retrieve.

pValue	[out] Pointer to variable that receives parameter value.

Return Values

If the function succeeds and *parameterId* specifies a N_TYPE_STRING type parameter, and *pValue* is NULL, the return value is length of the string (not including the NULL-terminator) *pValue* should point to.

If the function succeeds and *pValue* is not NULL, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*parameterId* specifies a non-static parameter and *hMatcher* is NULL. - or - *parameterId* specifies a non-N_TYPE_STRING type parameter and *pValue* is NULL.
N_E_PARAMETER	*parameterId* is invalid.

Remarks

The following values can be used for *parameterId*:

- FNCMP_COPYRIGHT
- FNCMP_MATCHING_THRESHOLD
- FNCMP_MAXIMAL_ROTATION
- FNCMP_MODE
- FNCMP_NAME
- FNCMP_VERSION_HIGH
- FNCMP_VERSION_LOW

To learn the type of the parameter pass value obtained with NParameterMakeId macro using N_PC_TYPE_ID code and the parameter id via *parameterId* parameter and pointer to NInt that will receive one of N_TYPE_XXX via *pValue* parameter. *hMatcher* can be NULL in this case.

See Also

FncMatcher Module | HFncMatcher | FncmSetParameter

5.5.1.6. FncmIdentifyEnd Function

Ends identification using the specified FncMatcher.

```
NResult N_API FncmIdentifyEnd(
        HFncMatcher hMatcher
);
```

Parameters

hMatcher	[in] Handle to the FncMatcher object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher* is NULL.
N_E_INVALID_OPERATION	Identification is not started.

See Also

FncMatcher Module | HFncMatcher | `FncmIdentifyStart` | `FncmIdentifyNext`

5.5.1.7. FncmIdentifyNext Function

Compares the specified packed NFRecord with the one identification was started with using the specified FncMatcher.

```
NResult N_API FncmIdentifyNext(
        HFncMatcher hMatcher,
        const void * templ,
        NSizeType templSize,
        FncmMatchDetails * pMatchDetails,
        NInt * pScore
);
```

Parameters

hMatcher	[in] Handle to the FncMatcher object.
templ	[in] Pointer to memory buffer containing

	packed NFRecord.
templSize	[in] Size of packed NFRecord.
pMatchDetails	[in, out] Pointer to FncmMatchDetails that is filled with details of the matching. Can be NULL.
pScore	[out] Pointer to NInt that receives similarity score.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher*, *templ*, or *pScore* is NULL.
N_E_END_OF_STREAM	*templSize* is less than expected.
N_E_FORMAT	Data in memory buffer *templ* points to is inconsistent with NFRecord format.
N_E_INVALID_OPERATION	Identification is not started.

Remarks

Value received via *pScore* is zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see FNCMP_MATCHING_THRESHOLD and FncmSetParameter function), and is greater than or equal to matching threshold otherwise.

If *pMatchDetails* is not NULL it should be a pointer obtained using FncmIdentifyStart function.

See Also

FncMatcher Module | HFncMatcher | FncmMatchDetails | FNCMP_MATCHING_THRESHOLD | FncmSetParameter | FncmIdentifyStart | FncmIdentifyEnd

5.5.1.8. FncmIdentifyStart Function

Starts identification with the specified packed NFRecord using the specified FncMatcher.

```
NResult N_API FncmIdentifyStart(
```

```
        HFncMatcher hMatcher,
        const void * templ,
        NSizeType templSize,
        FncmMatchDetails * * ppMatchDetails
);
```

Parameters

hMatcher	[in] Handle to the FncMatcher object.
templ	[in] Pointer to memory buffer containing packed NFRecord.
templSize	[in] Size of packed NFRecord.
ppMatchDetails	[in] Pointer to pointer to FncmMatchDetails that receives pointer to created match details. Can be NULL.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher* or *templ* is NULL.
N_E_END_OF_STREAM	*templSize* is less than expected.
N_E_FORMAT	Data in memory buffer *templ* points to is inconsistent with NFRecord format.
N_E_INVALID_OPERATION	Identification is already started.

Remarks

If *ppMatchDetails* is not NULL, the received pointer should be sequent passed to `FncmIdentifyNext` function to obtain details of the matching.

Finally match details should be deleted using `FncmMatchDetailsFree` function.

See Also

FncMatcher Module | HFncMatcher | FncmMatchDetails | `FncmMatchDetailsFree` | `FncmIdentifyNext` | `FncmIdentifyEnd`

5.5.1.9. FncmIsRegistered Function

Checks if FncMatcher library is registered.

```
NBool N_API FncmIsRegistered(void);
```

Return Values

NTrue if library is registered, NFalse otherwise.

See Also

FncMatcher Module

5.5.1.10. FncmMatchDetails Structure

Represents details of matching performed with FncMatcher.

```
typedef struct FncmMatchDetails_ { } FncmMatchDetails;
```

Fields

CenterX	X rotation point coordinate of the second (FncmVerify or FncmIdentifyNext) matched template.
CenterY	Y rotation point coordinate of the second (FncmVerify or FncmIdentifyNext) matched template.
MatedMinutiaCount	Number of mated minutiae in first and second matched NFRecords.
MatedMinutiae	Pointer to array of NIndexPair containing pairs of indexes of mated minutiae in first and second matched NFRecords.
Rotation	Rotation of second matched NFRecord against the first one.
Score	Similarity score of two matched NFRecords.
TranslationX	Horizontal translation of second matched NFRecord against the first one.
TranslationY	Vertical translation of second matched NFRecord against the first one.

See Also

FncMatcher Module

5.5.1.10.1. FncmMatchDetails.CenterX Field

X rotation point coordinate of the second (FncmVerify or FncmIdentifyNext) matched template.

```
NInt CenterX;
```

See Also

FncmMatchDetails

5.5.1.10.2. FncmMatchDetails.CenterY Field

Y rotation point coordinate of the second (FncmVerify or FncmIdentifyNext) matched template.

```
NInt CenterY;
```

See Also

FncmMatchDetails

5.5.1.10.3. FncmMatchDetails.MatedMinutiaCount Field

Number of mated minutiae in first and second matched NFRecords.

```
NInt MatedMinutiaCount;
```

See Also

FncmMatchDetails

5.5.1.10.4. FncmMatchDetails.MatedMinutiae Field

Pointer to array of NIndexPair containing pairs of indexes of mated minutiae in first and second matched NFRecords.

```
NIndexPair * MatedMinutiae;
```

See Also

FncmMatchDetails | NIndexPair

5.5.1.10.5. FncmMatchDetails.Rotation Field

Rotation of second matched NFRecord against the first one.

```
NByte Rotation;
```

Remarks

The rotation is specified in 180/128 degrees units in counterclockwise order.

To eliminate rotation, the second NFRecord minutiae and singular points have to be rotated by the value around center of its minutiae bounding box.

See Also

FncmMatchDetails

5.5.1.10.6. FncmMatchDetails.Score Field

Similarity score of two matched NFRecords.

```
NInt Score;
```

See Also

FncmMatchDetails

5.5.1.10.7. FncmMatchDetails.TranslationX Field

Horizontal translation of second matched NFRecord against the first one.

```
NInt TranslationX;
```

Remarks

To eliminate horizontal translation, the second NFRecord minutiae and singular points have to be shifted right by the value. Note that *`Rotation`* must be eliminated first.

See Also

FncmMatchDetails | *`Rotation`*

5.5.1.10.8. FncmMatchDetails.TranslationY Field

Vertical translation of second matched NFRecord against the first one.

```
NInt TranslationY;
```


Remarks

To eliminate horizontal translation, the second NFRecord minutiae and singular points have to be shifted down by the value. Note that *Rotation* must be eliminated first.

See Also

FncmMatchDetails | *Rotation*

5.5.1.11. FncmMatchDetailsFree Function

Deletes FncmMatchDetails.

```
void N_API FncmMatchDetailsFree(
        FncmMatchDetails * pMatchDetails
);
```

Parameters

pMatchDetails	[in] Pointer to FncmMatchDetails to delete.

Remarks

If *pMatchDetails* is NULL, does nothing.

See Also

FncMatcher Module | FncmMatchDetails

5.5.1.12. FncmReset Function

Sets default values for all parameters of the specified FncMatcher.

```
NResult N_API FncmReset(
        HFncMatcher hMatcher
);
```

Parameters

hMatcher	[in] Handle to FncMatcher object.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher* is NULL.

See Also

FncMatcher Module | HFncMatcher

5.5.1.13. FncmMatchDetailsSerialize Function

Serializes FncmMatchDetails structure into buffer.

```
NResult N_API FncmMatchDetailsSerialize(
        FncmMatchDetails * pMatchDetails,
        void * * pBuffer,
        NSizeType pBufferLength
);
```

Parameters

pMatchDetails	[in] Pointer to FncmMatchDetails structure to be serialized.
pBuffer	[out] Pointer to buffer where serialized matching details structre is put.
pBufferLength	[out] Returns the size of the buffer allocated for the serialized matching details.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*pMatchDetails* or *pBuffer* is NULL.
N_E_ARGUMENT	Data in *pMatchDetails* is inconsistent with FncmMatchDetails structure format.

Remarks

After the data in pBuffer is not neccessary anymore, this buffer must be free using NFree function.

See Also

FncMatcher Module | HFncMatcher | FncmMatchDetails

5.5.1.14. FncmSetParameter Function

Sets value of the specified parameter of the specified FncMatcher.

```
NResult N_API FncmSetParameter(
        HFncMatcher hMatcher,
        NUInt parameterId,
        const void * pValue
);
```

Parameters

hMatcher	[in] Handle to the FncMatcher object. Can be NULL if setting static parameter value.
parameterId	[in] Identifier of the parameter to set.
pValue	[in] Pointer to the parameter value to set.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT	Value *pValue* points to is invalid.
N_E_ARGUMENT_NULL	*parameterId* specifies a non-static parameter and *hMatcher* is NULL. - or - *pValue* is NULL.
N_E_ARGUMENT_OUT_OF_RANGE	Value *pValue* points to is out of range.
N_E_INVALID_OPERATION	*hMatcher* is not NULL and identification is started on the matcher.
N_E_PARAMETER	*parameterId* is invalid.

Error Code	Condition
N_E_PARAMETER_READ_ONLY	*parameterId* specifies read-only parameter.

Remarks

The following values can be used for *parameterId*:

- FNCMP_MATCHING_THRESHOLD
- FNCMP_MAXIMAL_ROTATION
- FNCMP_MODE

To learn the type of the parameter pass value obtained with NParameterMakeId macro using N_PC_TYPE_ID code and the parameter id via *parameterId* parameter and pointer to NInt that will receive one of N_TYPE_XXX via *pValue* parameter. *hMatcher* can be NULL in this case.

See Also

FncMatcher Module | HFncMatcher | FncmGetParameter

5.5.1.15. FncmVerify Function

Compares two packed NFRecords using the specified FncMatcher.

```
NResult N_API FncmVerify(
        HFncMatcher hMatcher,
        const void * template1,
        NSizeType template1Size,
        const void * template2,
        NSizeType template2Size,
        FncmMatchDetails * * ppMatchDetalis,
        NInt * pScore
);
```

Parameters

hMatcher	[in] Handle to the FncMatcher object.
template1	[in] Pointer to memory buffer containing first packed NFRecord.
template1Size	[in] Size of first packed NFRecord.
template2	[in] Pointer to memory buffer containing second packed NFRecord.

template2Size	[in] Size of second packed NFRecord.
ppMatchDetails	[in] Pointer to pointer to FncmMatchDetails that receives pointer to created match details. Can be NULL.
pScore	[out] Pointer to NInt that receives similarity score.

Return Values

If the function succeeds, the return value is N_OK.

If the function fails, the return value is one of the following error codes:

Error Code	Condition
N_E_ARGUMENT_NULL	*hMatcher*, *template1*, *template2*, or *pScore* is NULL.
N_E_END_OF_STREAM	*template1Size* or *template2Size* is less than expected.
N_E_FORMAT	Data in memory buffer *template1* or *template2* points to is inconsistent with NFRecord format.
N_E_INVALID_OPERATION	Identification is started.

Remarks

Value received via *pScore* is zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see FNCP_MATCHING_THRESHOLD and FncmSetParameter function), and is greater than or equal to matching threshold otherwise.

If *ppMatchDetails* is not NULL, the received pointer should be examined to obtain details of the matching.

Finally match details should be deleted using FncmMatchDetailsFree function.

See Also

FncMatcher Module | HFncMatcher | FncmMatchDetails | FNCMP_MATCHING_THRESHOLD | FncmSetParameter | FncmMatchDetailsFree | FncmIdentifyStart | FncmIdentifyNext | FncmIdentifyEnd

Chapter 6. Obsolete interface

6.1. Obsolete interface

The new FingerCell EDK comes with a new interface which is incompatible with the old one, for this reason the old interface is still suported by the EDK but will be dropped in the next version of the FingerCell EDK.

For old interface following library files must be used:

Windows

Import library: `FingerCell.dll.lib`.

DLL: `FingerCell.dll`.

Linux

Shared object: `libFingerCell.so`.

Requirements:

- `libNCore.so`.
- `NFRecord`.
- `FncMatcher`.
- `FncExtractor`.

6.2. Fingerprint images

Fingerprint image used by FingerCell library has to be an array of bytes of size width*height and pointer to the first element of this array has to be passed to libraries' functions. Lines of the image have to be stored in the array from top to bottom order. Next line must immediately follow the previous one (no padding). Each byte of the array corresponds to fingerprint image pixel (grayscale value). Value of 0 means black and value of 255 means white.

6.3. FingerCell library

FingerCell library is a fingerprint recognition engine that you can use in your embedded devices.

FingerCell library enables application to implement such scenarios as user enrollment, user verification and user identification using fingerprints. It provides a number of functions to implement such behavior.

When enrolling a user application can use features extraction functions that extracts features from fingerprint image (for more information see Fingerprint images and Features). Also fea-

tures generalization can be used to increase quality of the features. Then features can be stored in database for later access.

When verifying a user features that are extracted from fingerprint image are compared with etalon features that are in the database or somewhere else. See Verification.

When identifying a user features that are extracted from fingerprint image are compared with all features stored in the database until matching is successful or end of the database passed. See Identification.

Before using the library it has to be initialized. See Initialization and Contexts.

FingerCell library behavior is controlled through parameters.

6.3.1. Library functions

FingerCell library contains the following functions grouped by categories:

Registration	
VFRegistrationType	Returns registration type of FingerCell library
VFGenerateId	Generates registration id from serial number
VFRegister	Registers FingerCell library
Initialization	
VFInitialize	Initializes FingerCell library
VFFinalize	Uninitializes FingerCell library
Contexts	
VFCreateContext	Creates a context
VFFreeContext	Deletes the context
Parameters	
VFGetParameter	Retrieves parameter value

VFSetParameter	Sets parameter value
Features extraction	
VFExtract	Extracts features from fingerprint image
Features generalization	
VFGeneralize	Generalizes count features collections to single features collection
Verification	
VFVerify	Matches two features collections
Identification	
VFIdentifyStart	Starts identification with test features
VFIdentifyNext	Matches with sample features
VFIdentifyEnd	Ends identification

Each of these functions (except for the VFCreateContext) returns integer value to indicate result of the execution. If it is less than zero then execution of the function failed and the value means error code.

You can use `VFFailed` and `VFSucceeded` functions to determinate if the execution of the function failed or succeeded:

C:

```
#define VFFailed(result) ...
                #define VFSucceeded(result) ...
```

6.3.2. Error codes

The following error codes are defined:

General		
VFE_OK	0	OK, no error
VFE_FAILED	-1	Failed
VFE_OUT_OF_MEMORY	-2	Out of memory
VFE_NOT_INITIALIZED	-3	FingerCell library is not initialized
VFE_ARGUMENT_NULL	-4	One of the required function arguments is null
VFE_INVALID_ARGUMENT	-5	One of the function arguments has an invalid value
VFE_NOT_IMPLEMENTED	-9	Function is not implemented
Parameters		
VFE_INVALID_PARAMETER	-10	Parameter identifier is invalid (unknown)
VFE_PARAMETER_READ_ONLY	-11	Parameter is read only
Features extraction		
VFE_ILLEGAL_IMAGE_RESOLUTION	-101	Specified image resolution is illegal
VFE_ILLEGAL_IMAGE_SIZE	-102	Specified image size is illegal
VFE_LOW_QUALITY_IMAGE	-103	Warning. Image quaity is low
FingerCell specific		
VFE_INVALID_MODE	-1000	Function called in invalid mode

Features		
VFE_INVALID_FEATURES_FORMAT	-3000	Features passed to the function has invalid format

You can use VFErrorToString and VFResultToString functions to get string that describes error and result. VFCheckResult function throws exception in case of the function result indicates failure. These functions are not part of FingerCell library. For C they are implemented in FingerCell.h and FingerCellX.cpp files.

C:

```
string VFErrorToString(INT error);
string VFResultToString(INT result);
void VFCheckResult(INT result);
```

6.3.3. Registration

You have to register FingerCell library before using it. If library is not registered all functions (except for initialization, contexts, parameters and features functions) will return VFE_NOT_REGISTERED. There are several registration types available: not protected library, registration with HASP key, registration to PC, and registration in License Manager (LAN protection).

If you are using not protected library, you should use it directly without registration.

If you are using registration with HASP key, simply plug it to LPT or USB port before initializing FingerCell library.

If you are using registration to PC then call VFGenerateId function and pass serial number provided with your FingerCell library license. This function will generate registration id that you should send to Neurotechnologija (sales@neurotechnologija.com) or distributor from which library was acquired. Then pass serial number with received registration key to VFRegister function.

If you are using LAN protection then you must use string "LAN" as serial number and server name as registration key.

To determine how FingerCell library is registered (and if it needs registration at all) call VFRegistrationType function.

Example:

C:

```
// Registration to PC
```



```
// Your serial number here
CHAR serial_number[] = "xxxx-xxxx-xxxx-xxxx";

// Registration id generation
CHAR registration_id[100];
VFGenerateId(serial_number, registration_id);

// Received registration key
CHAR registration_key[] = "xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx";

// Register FingerCell library
VFRegister(serial_number, registration_key);
```

6.3.3.1. VFRegistrationType function

6.3.3.1. VFRegistrationType function

Returns FingerCell library registration type.

C:

```
INT VFINGER_API VFRegistrationType();
```

VF_RT_NOT_PROTECTED	0	FingerCell library is not protected. No need to register
VF_RT_HASP	1	HASP key found either on LPT or USB port. No need to register
VF_RT_PC	2	FingerCell library is registered to PC
VF_RT_UAREU	4	FingerCell library is registered to U.are.U scanner
VF_RT_LAN	8	FingerCell library is registered in License Manager on LAN
VF_RT_UNREGISTERED	6	FingerCell library is not registered. Call VFRegister function to register

6.3.3.2. VFGenerateId function

Generates registration id from specified serial number. Serial number and registration id have to be arrays of characters (strings) pointers to first element of each have to be passed to the function. Array for registration id has to be large enough to store the string (100 characters is enough).

C:

```
INT VFINGER_API VFGenerateId(CHAR * serial, CHAR * id);
```

Parameters:

[in]	serial, Serial	Serial number of FingerCell library license
[out]	id, Id	After execution of the function contains registration id for the serial number

Return values: If serial number or registration id is null returns `VFE_ARGUMENT_NULL`.If serial number is invalid returns `VFE_INVALID_SERIAL_NUMBER`.Otherwise generates registration id and returns `VFE_OK` (in case of error returns `VFE_FAILED`).

6.3.3.3. VFRegister function

Registers FingerCell library with specified serial number and registration key. Serial number and registration key have to be arrays of characters (strings) pointers to first element of each have to be passed to the function.

C:

```
INT VFINGER_API VFRegister(CHAR * serial, CHAR * key);
```

Parameters:

[in]	serial, Serial	Serial number of FingerCell library license. If you are using LAN protection then you must specify `"LAN"` as serial number.
[in]	key, Key	Registration key for serial number and registration id (received from Neurotechnologija or its distributor). If you are using LAN protection then you must specify server name as registration key.

Return values: If FingerCell library is not protected or already registered with HASP returns `VFE_REGISTRATION_NOT_NEEDED`.If serial number or registration key is null returns `VFE_ARGUMENT_NULL`.If serial number is invalid returns `VFE_INVALID_SERIAL_NUMBER`.Otherwise registers VeriFinger library and returns `VFE_OK` (in case of error returns `VFE_FAILED`)

6.3.4. Initialization

FingerCell library requires initialization to be performed before any function call and uninitialization to be performed after all function calls (except for contexts functions). This is performed using VFInitialize and VFFinalize functions.

Each successful call to `VFInitialize` should have a corresponding call to `VFFinalize`. So you can call `VFInitialize` more than one time, but you have to call `VFFinalize` equal number of times.

Also you may not call initialization functions at all if you will not work with default context, only with yours custom context.

See Contexts for more information.

Example:

C:

```
// Main application function
{
        // Application initialization code
        VFInitialize();
        // Other application code
        VFFinalize();
        // Application uninitialization code
}
```

6.3.4.1. VFInitialize function

Creates default context by calling VFCreateContext function and initialized FingerCell library.

C:

```
INT VFINGER_API VFInitialize();
```

Return values: If succeeded return value means number of times function have been called before. If it first call to the function return value will be zero.

6.3.4.2. VFFinalize function

Destroys default context by calling VFFreeContext function and uninitializes FingerCell lib-

rary if call to the function corresponds to first call to VFInitialize function.

C:

```
INT VFINGER_API VFFinalize();
```

Return values: Return value means number of times function should be more called (number of `VFInitialize` calls without `VFFinalize` calls).If FingerCell library was not initialized returns `VFE_NOT_INITIALIZED`.

6.3.5. Contexts

Context is a set of parameters and internal structures that FingerCell library functions use. They are created with VFCreateContext function and destroyed with VFFreeContext function.

Contexts enable different application parts to work with FingerCell library simultaneously. Inside one context no FingerCell functions should be called simultaneously because they are not guaranteed to be thread-safe. FingerCell functions called in different context are guaranteed to be thread-safe.

Parameters are set for the context. So you can use contexts not only to ensure that your application is thread safe, but to use different parameters in different situations also. For example you can perform features extraction for different scanners in different contexts with different set of parameters. For more information see Parameters.

Also particular FingerCell library functions should be called in particular order. If you have started identification (VFIdentifyStart) then you cannot call functions that work with internal matching structures (except for the VFIdentifyNext) such as VFSetParameter, VFGeneralize, VFVerify and VFIdentifyStart in the same context until you call VFIdentifyEnd. And you cannot call `VFIdentifyNext` and `VFIdentifyEnd` before you call `VFIdentifyStart` in the same context. In these situations functions will return `VFE_INVALID_MODE`.

Working from different threads:

C:

```
// First thread function
{
        // Create context
        HVFCONTEXT context = VFCreateContext();
        // Call FingerCell library functions, for example
        VFVerify(..., context);
        // Delete context
        VFFreeContext(context);
}
// Second thread function
{
        // Create context
        HVFCONTEXT context = VFCreateContext();
        // Call FingerCell library functions, for example
        VFIdentifyNext(..., context);
```

```
        // Delete context
        VFFreeContext(context);
}
```

Contexts with different parameters:

C:

```
HVFCONTEXT context1; // First context
HVFCONTEXT context2; // Second context
// Initialization function
{
        // Set parameters for default context
        VFSetParameter(..., NULL);
        VFSetParameter(..., NULL);
        // Create first context
        context1 = VFCreateContext();
        // Set parameters for first context
        VFSetParameter(..., context1);
        VFSetParameter(..., context1);
        // Create second context
        context2 = VFCreateContext();
        // Set parameters for second context
        VFSetParameter(..., context2);
        VFSetParameter(..., context2);
}

// Some application function
{
        HVFCONTEXT context;
        if (/* image from first scanner */)
                context = context1;
        else if(/* image from second scanner */)
                context = context2;
        else
                context = NULL; // default context
        // Call FingerCell library functions, for example
        VFExtract(..., context);
}

// Uninitialization function
{
        // Delete first context
        VFFreeContext(context1);
        // Delete second context
        VFFreeContext(context2);
}
```

Example: Wrong functions call order:

C:

```
// Some application function
```

```
{
        //...
        VFIdentifyStart(...);
        for (...)
        VFIdentifyNext(...);
        VFVerify(...); // Error, returns VFE_INVALID_MODE
        VFIdentifyEnd(...);
        //...
        VFExtract(...);
        VFIdentifyNext(...); // Error, returns VFE_INVALID_MODE
}
```

6.3.5.1. VFCreateContext function

Creates context with default parameters.

C:

```
HVFCONTEXT VFINGER_API VFCreateContext();
```

Return values: Return value is newly created context.If context cannot be created returns `VFE_OUT_OF_MEMORY`.

6.3.5.2. VFFreeContext function

Deletes context created with VFCreateContext.

C:

```
INT VFINGER_API VFFreeContext(HVFCONTEXT context);
```

Parameters:

	context, Context	Context to delete

Return values: If context is null returns `VFE_ARGUMENT_NULL` else returns `VFE_OK`.

6.3.6. Parameters

Some FingerCell algorithm aspects are controlled through parameters. Parameters are retrieved and set for the specified context by VFGetParameter and VFSetParameter functions. Some parameters are read only (informational). If you will try to set a read only parameter `VFSetParameter` function will return `VFE_PARAMETER_READ_ONLY`. If you will pass an invalid parameter identifier to one of these functions it will return `VFE_INVALID_PARAMETER`.

Parameters can be of the following types:

Referenced as	Size (bytes)	VF_TYPE_XXX constant	C equivalent
Void		VF_TYPE_VOID0	
Byte	1	VF_TYPE_BYTE1	BYTE
Signed byte	1	VF_TYPE_SBYTE2	SBYTE
Word	2	VF_TYPE_WORD3	WORD
Short integer	2	VF_TYPE_SHORT4	SHORT
Double word	4	VF_TYPE_DWORD5	DWORD
Integer	4	VF_TYPE_INT6	INT
Boolean	4	VF_TYPE_BOOL10	BOOL
Char	1	VF_TYPE_CHAR20	CHAR
String	4	VF_TYPE_STRING100	CHAR*

To determine parameter type call VFGetParameter function with parameter identifier VFP_TYPE and value - needed parameter identifier. Also you may use VFGetParameterType function. Result of the function will be one of VF_TYPE_XXX constants.

When retrieving a parameter value pass pointer to variable of parameter type as value for VFGetParameter function.

For string parameter pass pointer to first char in the string as value. To retrieve length of the string (not including the terminating null character) pass null as value. Function will return length of the string.

When setting a parameter value pass the value casted to double word to VFSetParameter function.

There are functions VFGetXxxParameter and VFSetXxxParameter that work with particular

parameter type.

For general parameters there are special functions VFGetXxx defined.

The following parameter identifiers are defined (grouped by categories):

Identifier	Value	Read only	Type	Description
General				
VFP_TYPE	0	x		See parameters types earlier
VFP_NAME	10	x	String	Name of the FingerCell library
VFP_VERSION_HIGH	11	x	Double word	Major version of FingerCell library
VFP_VERSION_LOW	12	x	Double word	Minor version of FingerCell library
VFP_COPYRIGHT	13	x	String	Copyright of FingerCell library
Features extraction				
VFP_EXTRACT_FEATURES	110		Integer	Obsolete, will be removed in the next version
VFP_RETURNED_IMAGE	10002		Integer	Specifies what image features extraction function will return. Can be one of the following:
VF_RETURNED_IMAGE_NONE	0	None image is returned - the image will be the same as passed to features extraction function		
VF_RETURNED_IMAGE_BINARIZED	100	Binarized image will be returned (default)		
VF_RETURNED_IMAGE_SKELETONIZED	200	Skeletonized image will be returned		

Features matching (Verification and Identification)				
VFP_MATCHING_THRESHOLD	200		Integer	Minimal similarity of two features collections that are identical. Must be not less that zero. See also Matching threshold
VFP_MAXIMAL_ROTATION	201		Integer	Maximal rotation of two features collection to each other. Must be in range VFDIR_0..VFDIR_180. See also information about directions in Features and Matching details
VFP_MATCH_FEATURES	210		Integer	Obsolete, will be removed in the next version
VFP_MATCHING_SPEED	220		Integer	Speed of features matching. Can be one of the following:
VF_MATCHING_SPEED_LOW	0	Low matching speed		
VF_MATCHING_SPEED_HIGH	256	High matching speed		
Features generalization				
VFP_GENERALIZATION_THRESHOLD.	300		Integer	Has the same meaning for features generalization as VFP_MATCHING_THRESHOLD parameter for features matching. See also Matching threshold
VFP_GEN_MAXIMAL_ROTATION	201		Integer	Has the same meaning for features generalization as VF_MAXIMAL_ROTATION parameter for features matching

FingerCell specific				
VFP_MODE	1000		Integer	Specifies mode in which FingerCell algorithm is operating (optimized parameter set) Can be one of the following:
VF_MODE_GENERAL	0	General		
VF_MODE_DIGITALPERSONA_UAREU	100	DigitalPersona U.are.U		
VF_MODE_BIOMETRIKA_FX2000	200	BiometriKa FX2000		
VF_MODE_KEYTRONIC_SECUREDESKTOP	300	Keytronic SecureDesktop		
VF_MODE_IDENTIX_TOUCHVIEW	400	Identix TouchView		
VF_MODE_PRECISEBIOMETRICS_100CS	500	PreciseBiometrics 100CS		
VF_MODE_STMICROELECTRONICS_TOUCHCHIP	600	STMicroelectronics TouchChip		
VF_MODE_IDENTICATORTECHNOLOGY_DF90	700	IdenticatorTechnology DF90		
VF_MODE_AUTHENTEC_AFS2	800	Authentec AFS2		
VF_MODE_AUTHENTEC_AES4000	810	Authentec AES4000		
VF_MODE_ATMEL_FINGERCHIP	900	Atmel FingerChip		

VF_MODE_BMF_BLP100	1000	BMF BLP100
VF_MODE_SECUGEN_HAMSTER	1100	SecuGen Hamster
VF_MODE_ETHENTICA	1200	Ethentica
VF_MODE_CROSSMATCH_VERIFIER300	1300	CrossMatch Verifier 300

6.3.6.1. VFGetParameter function

Retrieves specified parameter value for specified context.

C:

```
INT VFINGER_API VFGetParameter(INT parameter, VOID * value, HVFCONTEXT context);
```

Parameters:

	parameter, Parameter	Parameter identifier to retrieve
[out]	value, Value	Pointer to variable that will receive parameter value
	Context, Context	Context to retrieve parameter from. Null for default context

Return values: If context is null and FingerCell library is not initialized returns VFE_NOT_INITIALIZED.If parameter is invalid (unknown) returns VFE_INVALID_PARAMETER.If value is null returns VFE_ARGUMENT_NULL. For string parameters returns length of the string (not including the terminating null character).Otherwise returns VFE_OK.

Example:

C:

```
// Some application function
{
        CHAR *name;
        INT l;
        DWORD version;
```

```
        INT vfp_mode_type;
        INT mode;

        // Get FingerCell library name
        l = VFGetParameter(VFP_NAME, NULL, NULL);
        name = (CHAR*)malloc((l + 1) * sizeof(CHAR));
        VFGetParameter(VFP_NAME, name, NULL);
        printf(name);
        free(name);

        // Get FingerCell library major version
        VFGetParameter(VFP_VERSION_HIGH, version, NULL);
        printf("Version: %u.%u", version, HIWORD(version),
        LOWORD(version));

        // Determine parameter VFP_MODE type
        vfp_mode_type = VFGetParameter(VFP_TYPE, (VOID*)VFP_MODE, NULL);

        // returned value: VF_TYPE_INT
        // Get integer parameter VFP_MODE value
        VFGetParameter(VFP_MODE, &mode, NULL);
        printf("Mode: %d", mode);
}
```

6.3.6.2. VFSetParameter function

Sets specified parameter value for specified context.

C:

```
INT VFINGER_API VFSetParameter(INT parameter, DWORD value, HVFCONTEXT context);
```

Parameters:

	parameter, Parameter	Parameter identifier to set
	value, Value	Parameter value to set
	context, Context	Context to set parameter to. Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is started returns `VFE_INVALID_MODE`.If parameter is invalid (unknown) returns `VFE_INVALID_PARAMETER`.Otherwise returns `VFE_OK`.

Example:

C:

```
// Some application function
{
        // Set VFP_EXTRACT_FEATURES parameter to VF_ALL_FEATURES
        VFSetParameter(VFP_EXTRACT_FEATURES, (DWORD) VF_ALL_FEATURES, NULL);
}
```

6.3.6.3. Additional functions

The following functions are not part of FingerCell library. They are implemented in `FingerCellX.h` and `FingerCellX.cpp` files.

`VFGetXxxParameter` functions retrieve parameters values of some type.

`VFSetXxxParameter` functions set parameters values of some type.

`VFGetXxx` functions retrieve general parameters values (`VFP_TYPE`, `VFP_NAME`, `VFP_VERSION_HIGH`, `VFP_VERSION_LOW`, `VFP_COPYRIGHT`).

C:

```
// Get
BYTE VFGetByteParameter(INT parameter, HVFCONTEXT context = NULL);
SBYTE VFGetSByteParameter(INT parameter, HVFCONTEXT context = NULL);
WORD VFGetWordParameter(INT parameter, HVFCONTEXT context = NULL);
SHORT VFGetShortParameter(INT parameter, HVFCONTEXT context = NULL);
DWORD VFGetDWordParameter(INT parameter, HVFCONTEXT context = NULL);
INT VFGetIntParameter(INT parameter, HVFCONTEXT context = NULL);
bool VFGetBoolParameter(INT parameter, HVFCONTEXT context = NULL);
TCHAR VFGetCharParameter(INT parameter, HVFCONTEXT context = NULL);
string VFGetStringParameter(INT parameter, HVFCONTEXT context = NULL);
// Set
void VFSetByteParameter(INT parameter, BYTE value, HVFCONTEXT context = NULL);
void VFSetSByteParameter(INT parameter, SBYTE value, HVFCONTEXT context = NULL);
void VFSetWordParameter(INT parameter, WORD value, HVFCONTEXT context = NULL);
void VFSetShortParameter(INT parameter, SHORT value, HVFCONTEXT context = NULL);
void VFSetDWordParameter(INT parameter, DWORD value, HVFCONTEXT context = NULL);
void VFSetIntParameter(INT parameter, INT value, HVFCONTEXT context = NULL);
void VFSetBoolParameter(INT parameter, bool value, HVFCONTEXT context = NULL);
void VFSetCharParameter(INT parameter, TCHAR value, HVFCONTEXT context = NULL);
void VFSetStringParameter(INT parameter,
        const string& value, HVFCONTEXT context = NULL);
// Get general
INT VFGetParameterType(INT parameter);
string VFGetName();
DWORD VFGetVersionHigh();
DWORD VFGetVersionLow();
string VFGetCopyright();
```

Parameters:

	parameter, Parameter	Parameter identifier to retrieve or set
	value, Value	Parameter value to set
	context, Context	context retrieve or set parameter value for (by default - null, default context)

Return values: `VFGetXxxParameter` functions return specified parameter value.`VFGetXxx` functions return corresponding general parameter value.Other functions return nothing.

Exceptions: All functions raise exception in case of error.

Example:

C:

```
// Some application function
{
        string name;
        DWORD version;
        INT vfp_mode_type;
        INT mode;

        // Get FingerCell library name
        name = VFGetStringParameter(VFP_NAME);
        // or
        name = VFGetName();
        printf(name);
        // Get FingerCell library major version
        version = VFGetDWordParameter(VFP_VERSION_HIGH);
        // or
        version = VFGetVersionHigh();
        printf("Version: %u.%u", version, HIWORD(version),
        LOWORD(version));

        // Determine parameter VFP_MODE type
        vfp_mode_type = VFGetParameterType(VFP_MODE);
        // returned value: VF_TYPE_INT

        // Get integer parameter VFP_MODE value
        mode = VFGetIntParameter(VFP_MODE);
        printf("Mode: %d", mode);
}
```

6.3.7. Features extraction

You can use features extraction to extract features from fingerprint image and then store them in a database (enroll fingerprint). For more information see Fingerprint images and Features.

Use VFExtract function to perform features extraction.

6.3.7.1. VFExtract function

Extracts features from fingerprint image in the specified context.

Image has to be an array of bytes of size width*height and pointer to the first element of this array has to be passed to this function. Image resolution has to be passed to the function. Function resizes image to resolution of `VF_IMAGE_RESOLUTION` dpi (dots per inch) internally. Filtered image that is returned by the function is resized back to original resolution. Features returned by the function have resolution of `VF_IMAGE_RESOLUTION` dpi, so if you wish to display features on the image you have to draw features on the image resized to `VF_IMAGE_RESOLUTION` dpi.

Features have to be an array of bytes and pointer to the first element of the array has to be passed to this function (in Visual Basic case - image array are passed to functions). Number of bytes occupied by features in the array will be returned by the function in size (Size) parameter. If array size is less than needed then behavior of the function is undefined. To ensure that array is large enough set its size to at least `VF_MAX_FEATURES_SIZE`. For more information see Features.

The function uses features extraction and FingerCell specific parameters.

C:

```
#define VF_IMAGE_RESOLUTION 250
#define VF_FEATURES_RESOLUTION 500
#define VF_MAX_FEATURES_SIZE 10000
INT VFINGER_API VFExtract(INT width, INT height, BYTE * image,INT resolution,
        BYTE * features, DWORD * size, HVFCONTEXT context);
```

Parameters:

	width, Width	Fingerprint image width. After resize have to be in range from `VF_MIN_IMAGE_DIMENSION` to `VF_MAX_IMAGE_DIMENSION`
	height, Height	Fingerprint image height. After resize have to be in range from `VF_MIN_IMAGE_DIMENSION` to `VF_MAX_IMAGE_DIMENSION`
[in/out]	image, Image	Fingerprint image to extract features from. After execution of the function contains filtered image

	resolution, Resolution	Resolution of the fingerprint image (in dots per inch). Must be in range from VF_MIN_IMAGE_RESOLUTION to VF_MAX_IMAGE_RESOLUTION
[out]	features, Features	After execution of the function contains features extracted from fingerprint image
[out]	size, Size	After execution of the function contains size of the features in bytes.
	context, Context	Context to perform features extraction in. Null for default context

Return values: If context is null and FingerCell library is not initialized returns VFE_NOT_INITIALIZED.If fingerprint image width or height is not in legal range returns VFE_ILLEGAL_IMAGE_SIZE.If resolution is not in legal range returns VFE_ILLEGAL_IMAGE_RESOLUTION.If image, features or size is null returns VFE_ARGUMENT_NULL.Otherwise performs features extraction and returns either VFE_OK or VFE_LOW_QUALITY_IMAGE (if fingerprint image quality is low). VFE_LOW_QUALITY_IMAGE is only a warning. Calling application can either ignore it or ask the user to rescan the fingerprint.

Example:

C:

```
// Extraction function
{
        INT width, height;
        BYTE *image;
        INT resolution;
        BYTE features[VF_MAX_FEATURES_SIZE];
        DWORD size;

        // Load the image from file or get the image from scanner
        // ...
        VFExtract(width, height, image, resolution, features, &size, NULL);
}
```

6.3.8. Features generalization

You can use features generalization to increase quality of the recognition. Generalization performs conjunction of several features collections to one collection, validates each feature and removes noisy features. You can use features generalization during enrollment. To obtain fea-

tures for generalization use features extraction functions.

Generalization uses VF_GENERALIZATION_THRESHOLD parameter for matching to determine if provided features collections are of the same finger. For more information see Matching threshold.

Use VFGeneralize function to perform features generalization.

6.3.8.1. VFGeneralize function

Performs generalization of features collections in the specified context. Currently generalization can be performed only for VF_GENERALIZE_COUNT features collections.

This function uses features extraction, features generalization, features matching and FingerCell specific parameters.

C:

```
#define VF_GENERALIZE_COUNT 3
INT VFINGER_API VFGeneralize(INT count, const BYTE * const *gen_features,
        BYTE * features, DWORD * size, HVFCONTEXT context);
```

Parameters:

	count, Count	Count of features collections to generalize
[in]	gen_features, GenFeatures	Array of features collections to generalize
[out]	features, Features	After execution of the function contains generalized features
[out]	size, Size	After execution of the function contains size of generalized features in bytes.
	context, Context	Context to perform features generalization in. Null for default context

Return values: If context is null and FingerCell library is not initialized returns VFE_NOT_INITIALIZED.If identification is started returns VFE_INVALID_MODE.If count of features collections is other that VF_GENERALIZE_COUNT returns VFE_INVALID_ARGUMENT.If features collections are null returns VFE_ARGUMENT_NULL.If one of the passed features collections has invalid format returns VFE_INVALID_FEATURES_FORMAT.If features collections cannot be generalized returns

VFE_FAILED.Otherwise return index of features collection on which base features generalization has been performed.

Example:

C:

```
// Generalization function
{
        BYTE *feats[3];
        BYTE features[VF_MAX_FEATURES_SIZE];
        DWORD size;
        feats[0] = /*obtain first fingerprint features*/;
        feats[1] = /*obtain second fingerprint features*/;
        feats[2] = /*obtain third fingerprint features*/;
        if (VFSucceeded(VFGeneralize(3, feats, features, &size, NULL))
                printf("Generalization succeeded");
        else
                printf("Generalization failed");
}
```

6.3.9. Verification

You can use verification to determinate if two features collections are of the same finger. It uses VFP_MATCHING_THRESHOLD parameter (See Matching threshold). To obtain features from fingerprint image use features extraction functions. Also you may use features generalization functions to increase recognition reliability.

Use VFVerify function to perform verification.

6.3.9.1. VFVerify function

Performs two features collections verification in the specified context.

Pass pointer to matching details structure that will receive details of features collections matching (set size (Size) member before calling the function to actual size of the structure). Pass null if you are not interested in matching details. For more information see Matching details.

This function uses features matching and FingerCell specific parameters. For more information see Parameters.

C:

```
INT VFINGER_API
VFVerify(const BYTE * features1, const BYTE * features2,
        VFMATCHDETAILS * md, HVFCONTEXT context);
```

Parameters:

[in]	features1, Features1	First fingerprint features
[in]	features2, Features2	Second fingerprint features
[in/out]	md, MD	After execution of the function contains details of features collections matching
	context, Context	Context to perform verification in. Null for default context

Return values: If context is null and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is started returns `VFE_INVALID_MODE`.If one of the features collections is `null` returns `VFE_ARGUMENT_NULL`.If one of the passed features collections has invalid format returns `VFE_INVALID_FEATURES_FORMAT`.If insufficient memory then returns `VFE_OUT_OF_MEMORY`.If features collections similarity is high enough (see `VFP_MATCHING_THRESHOLD` in Parameters) returns `VFE_OK` (the same finger features collections). Otherwise returns `VFE_FAILED`.

Example:

C:

```
// Verification function
{
        BYTE *features1, *features2;
        VFMATCHDETAILS md;
        BOOL result;
        features1 = /*obtain first fingerprint features*/;
        features2 = /*obtain second fingerprint features*/;
        md.size = sizeof(md);
        result = VFSucceeded(VFVerify(features1, features2, &md, NULL));
        if (result)
                printf("Same finger. Similarity: %d", md.similarity);
        else
                printf("Different fingers. Similarity: %d", md.similarity);
}
```

6.3.10. Identification

Use identification to identify fingerprint in the database.

First start identification with unknown fingerprint (test) features. Use VFIdentifyStart function.

Then walk through all database features (sample features) and match them with test features

(with VFIdentifyNext function) until matched (function returns VFE_OK) or end of the database passed. It uses VFP_MATCHING_THRESHOLD parameter (see Matching threshold).

You may also use G to increase speed of the identification: match first sample features which G is equal to test features G; then sample features which G difference with test features is 1, then with G difference 2 and so on. It is a quite high probability that fingerprint will be identified during first matches if it is in the database. See also Features.

End the identification (VFIdentifyEnd function).

To obtain features for identification use features extraction function (for enrollment in the database you may also use features generalization functions).

Example:

C:

```
// Identification function
{
        BYTE *test_features;
        BYTE *sample_features;
        BOOL found;

        test_features = /*obtain features of fingerprint to identify*/;
        VFIdentifyStart(test_features, NULL);
        found = FALSE;
        for (/* walk through database */)
        {
                sample_features = /*some features from the database*/;
                if (VFSucceeded(VFIdentifyNext(sample_features, NULL, NULL)))
                {
                        found = TRUE;
                        break;
                }
        }
        VFIdentifyEnd(NULL);
        if (found)
                printf("Fingerprint found in the database");
        else
                printf("Fingerprint not found in the database");
}
```

6.3.10.1. VFIdentifyStart function

Starts identification with specified test features in specified context.

This function uses features matching and FingerCell specific parameters. For more information see Parameters.

C:

```
INT VFINGER_API VFIdentifyStart(const BYTE * test_features, HVFCONTEXT context);
```

Parameters:

[in]	test_features, TestFeatures	Test features
	context, Context	Context to start identification in Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is started returns `VFE_INVALID_MODE`.If test features collection has invalid format returns `VFE_INVALID_FEATURES_FORMAT`.If test features are null returns `VFE_ARGUMENT_NULL`.If insufficient memory then returns `VFE_OUT_OF_MEMORY`.

6.3.10.2. VFIdentifyNext function

Matches sample features with test features in specified context.

Pass pointer to matching details structure that will receive details of features collections matching (set size (Size) member before calling the function to actual size of the structure). Pass null if you are not interested in matching details. For more information see Matching details.

This function uses features matching and FingerCell specific parameters. For more information see Parameters.

C:

```
INT VFINGER_API VFIdentifyNext(const BYTE * sample_features,
        VFMATCHDETAILS * md, HVFCONTEXT context);
```

Parameters:

[in]	sample_features, SampleFeatures	Sample features
[in/out]	md, MD	After execution of the function contains details of features collections matching.
	Context, Context	Context to perform features matching in. Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is not started returns `VFE_INVALID_MODE`.If sample features are null returns `VFE_ARGUMENT_NULL`.If test features collection has invalid format returns `VFE_INVALID_FEATURES_FORMAT`.If features collections similarity is high enough (see `VFP_MATCHING_THRESHOLD` in Parameters) returns `VFE_OK` (the same finger features collections). Otherwise returns `VFE_FAILED`.

6.3.10.3. VFIdentifyEnd function

Ends the identification started with VFIdentifyStart function in specified context.

C:

```
INT VFINGER_API VFIdentifyEnd(HVFCONTEXT context);
```

Parameters:

	context, Context	Context to end identification in. Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is not started returns `VFE_INVALID_MODE`.

6.3.11. Matching threshold and similarity

FingerCell features matching algorithm provides value of features collections similarity as a result. It can be obtained in matching details. The higher is similarity, the higher is probability that features collections are obtained from the same finger fingerprints.

You can set matching threshold - the minimum similarity value that verification and identification functions accept for the same finger fingerprints. You can set the matching threshold using `VFP_MATCHING_THRESHOLD` parameter (`VFP_GENERALIZATION_THRESHOLD` for features generalization).

Matching threshold is linked to false acceptance rate (FAR, different fingers fingerprints erroneously accepted as the of the same finger) of matching algorithm. The higher is threshold, the lower is FAR and higher FRR (false rejection rate, same finger fingerprints erroneously accepted as different fingers fingerprints) and vice a versa. You can use `VFMatchingThresholdToFAR` and `VFFARToMatchingThreshold` functions to convert, matching threshold to FAR in percents and vice a versa. Only values of and for FAR between 1% and 0.001% can be calculated more or less correctly. All other values are calculated very approximately. These functions are not part of FingerCell library; they are implemented in `FingerCellX.h` and `FingerCellX.cpp` files.

C:

```
double VFMatchingThresholdToFAR(INT th);
```



```
INT VFFARToMatchingThreshold(double f);
```

6.3.12. Matching details

Matching details describes relationship between two features collections determined during verification or identification. Matching details are defined as structure, pointer to which you can pass to verification or identification functions. It can be one of the following structures. When passing to the function set size (Size) member to size of actual structure and cast pointer to the structure to pointer to the first structure. See also Features.

C:

```
#define VF_MAX_MINUTIA_COUNT 1024
typedef struct _VFMATCHDETAILS
{
        DWORD size;
        INT similarity;
        INT rotation;
        INT trans_x;
        INT trans_y;
} VFMATCHDETAILS;

typedef struct _VFMATCHDETAILSEX
{
        DWORD size;
        INT similarity;
        INT rotation;
        INT trans_x;
        INT trans_y;
        INT mm_count;
        SHORT mm[2][VF_MAX_MINUTIA_COUNT];
} VFMATCHDETAILSEX;
```

Members:

size, Size	Size of the structure. Set this member before calling a function
similarity, Similarity	Two features collections similarity value. The bigger is value the higher is similarity. See also Matching threshold and similarity
rotation, Rotation	Rotation of two features collections to each other. It is an angle in range [`0`, `VFDIR_360`). See also information about directions in Features
Trans_x, TransX	Translation between two features collections along X axis

Trans_y, TransY	Translation between two features collections along Y axis
mm_count, MMCount	Count of minutiae common for two features collections in matched minutiae array
mm, MM	Matched minutiae array (common minutiae for two features collections). First index of the array means first (0) or second (1) features collection, second - index of minutia in features collection. See also Features

Example:

C:

```
//Identification function
{
        BYTE *test_features;
        BYTE *sample_features;
        VFMATCHDETAILS md;

        //...
        VFIdentifyStart(test_features, NULL);
        md.size = sizeof(md);
        for (...)
        {
                VFIdentifyNext(sample_features, &md, NULL);
                printf("Similarity: %d", md.similarity);
        }
        VFIdentifyEnd(NULL);
}

// Verification function
{
        BYTE *features1, *features2;
        VFMATCHDETAILSEX md;
        INT I;

        // ...
        md.size = sizeof(md);
        VFVerify(features1, features2, (VFMATCHDETAILS)&md, NULL);
        for (i = 0; i < md.mm_count; i++)
                printf("Matched minutia %d: index in first features - %d"
                "; index in second features - %d\n", i, md.mm[0][i], md.mm[1][i]);
}
```

6.3.13. Fingerprint features

Features or features collection or template are data extracted from fingerprint image that is

used in verification and identification. To obtain features from fingerprint image use features extraction functions. You may also use features generalization to improve quality of the features.

Features are stored in array of bytes. Size of the array will never exceed `VF_MAX_FEATURES_SIZE`.

You can use `VFDecompressFeatures` and `VFCompressFeatures` functions to decompress features to and compress features from structure. These functions are not part of FingerCell library. They are implemented in `FingerCellX.h` and `FingerCellX.cpp` files. Also you may use `CVFFeatures` class (`VFFeatures.h` and `VFFeatures.cpp` files in sample application) that encapsulates features, compression and decompression. You can use the class or compression and decompression functions to implement manual features editing in your application.

C:

```
typedef enum _VFSingularPointType
{
        vfsptUnknown = 0,
        vfsptCore = 1,
        vfsptDoubleCore = 2,
        vfsptDelta = 3
} VFSingularPointType;

typedef struct _VFSingularPoint
{
        INT X;
        INT Y;
        VFSingularPointType T;
        BYTE D;
} VFSingularPoint;

typedef enum _VFMinutiaType
{
        vfmtUnknown = 0,
        vfmtEnd = 1,
        vfmtBifurcation = 2
} VFMinutiaType;

typedef struct _VFMinutia
{
        INT X;
        INT Y;
        VFMinutiaType T;
        BYTE D;
        BYTE C;
        BYTE G;
} VFMinutia;

// Features compression
INT VFINGER_API VFFeatSet(BYTE g, INT mCount, const VFMinutia * m,
        INT spCount, const VFSingularPoint * sp, INT boWidth, INT boHeight,
        const BYTE * bo, BYTE * features);
```

```
// Features decompression
INT VFINGER_API VFFeatGetG(const BYTE * features);
INT VFINGER_API VFFeatGetMinutiaCount(const BYTE * features);
INT VFINGER_API VFFeatGetMinutiae(const BYTE * features, VFMinutia * m);
INT VFINGER_API VFFeatGetSPCount(const BYTE * features);
INT VFINGER_API VFFeatGetSP(const BYTE * features, VFSingularPoint * sp);
INT VFINGER_API VFFeatGetBOSize(const BYTE * features,
INT * pWidth, INT * pHeight);
INT VFINGER_API VFFeatGetBO(const BYTE * features, BYTE * bo);
```

All functions take features (Features) argument as compressed features and/or m (M) argument as minutia array (mCount (MCount) is length of the array), sp (SP) argument as singular point array (spCount (SPCount) is length of the array), bo (BO) argument as blocked orientation image (in similar format as fingerprint image; boWidth (BOWidth) and boHeight (BOHeight) are accordingly width and height of blocked orientations image). `VFFeatSet` function returns size of compressed features.

Features consist of:

- G - obtained using `VFFeatGetG` function
- Minutiae - obtained using `VFFeatGetMinutiae` function (number of minutiae obtained using `VFGetMinutiaCount` function)
- Singular points - obsolete, obtained using `VFFeatGetSP` function (number of singular points obtained using `VFGetSPCount` function). FingerCell does not extract singular points
- Blocked orientations - obsolete, obtained using `VFFeatGetBO` function (size of blocked orientations obtained using `VFGetBOSize` function). FingerCell does not extract blocked orientations

G is a fingerprint global feature that reflects ridge density. It can have values from 0 to 255, so it occupies one byte. The bigger is value the bigger is ridge density. You can use `VFFeatG` function to get ridge density.

Minutiae are points in fingerprint image where finger ridges end or separate. Each minutia is a structure with the following members: X, Y - coordinates in pixels, T - type, D - direction, C - curvature, G - g.

Singular points are points in fingerprints image where finger ridges screw. Each singular point is a structure with the following members: X, Y - coordinates in pixels, T - type, D - direction.

Each minutia and singular point has x and y coordinates (from top-left corner of the image) and direction.

Direction is value in range [`VFDIR_0`, `VFDIR_360`) - byte value. To convert it to degrees

multiply by 180 and divide by VFDIR_180 and vice a versa to convert degrees to direction. Also you may use VFDirToDeg and VFDegToDir functions. To covert them to radians and vice a versa use VFDirToRad and VFRadToDir functions. The following constants are defined:

VFDIR_0	0	0°
VFDIR_45	30	45°
VFDIR_90	VFDIR_45 * 2	90°
VFDIR_135	VFDIR_45 * 3	135°
VFDIR_180	VFDIR_45 * 4	180°
VFDIR_225	VFDIR_45 * 5	225°
VFDIR_270	VFDIR_45 * 6	270°
VFDIR_315	VFDIR_45 * 7	315°
VFDIR_360	VFDIR_45 * 8	360°
VFDIR_UNKNOWN	127	Unknown direction
VFDIR_BACKGROUND	255	Background

Blocked orientations are fingerprint image divided in blocks of VF_BLOCK_SIZE * VF_BLOCK_SIZE pixels. Value of each block is ridges orientation of fingerprint image in that block. Can be in range [VFDIR_0, VFDIR_180) or VFDIR_UNKNOWN or VFDIR_BACKGROUND - byte value.

C:

```
// Conversion from VFDIR_XXX to degrees and vice a versa
#define VFDirToDeg(dir) ...
#define VFDirToDegF(dir) ...
#define VFDegToDir(deg) ...

// Conversion from VFDIR_XXX to radians and vice a versa
#define VFDirToRad(dir) ...
#define VFRadToDir(a) ...
```



```
// Orientation stuff
#define VFIsBadArea(orient) ...
#define VFIsGoodArea(orient) ...
#define VFTheOrient(orient) ...
#define VFIsUnknown(orient) ...
#define VFIsOrient(orient) ...
```


Appendix A. Support

Neurotechnologija provides customer support during the entire period, while the customer develops and uses his own system based on our products. Customers are welcome to contact:

- <support@neurotechnologija.com> for any help on solving the other development problems.

Appendix B. Distribution Content

FingerCell EDK distribution contains the following folders and files:

B.1. bin

Subdirectories of this directory contain shared libraries ,binary files of demo applications, and activation files for corresponding operating systems.

`linux_arm`	Contains FingerCell EDK binary files for linux OS (platform arm).
`ppc03_armv4`	Contains FingerCell EDK binary files for MS Windows CE (platform arm).
`ppc03_ipaq`	Contains FingerCell EDK binary files for MS Windows CE (platform arm) users using ipaq 5500 series.

B.2. documentation

Contains comprehensive documentation of the FingerCell EDK and the license.

B.3. include

Subdirectories of this directory contains include files for corresponding operating systems.

`linux`	Contains FingerCell EDK include files for linux OS (platform arm).
`Windows`	Contains FingerCell EDK include files for MS Windows CE OS (platform arm 32bit).

B.4. lib

Subdirectories of this directory contains library files for corresponding operating systems.

`linux_arm`	Contains FingerCell EDK library files for linux OS (platform arm) .
`ppc03_armv4`	Contains FingerCell EDK library files for MS Windows CE OS (platform arm) .

B.5. samples

Subdirectories of this directory contains source code of demo applications for corresponding platforms.

`linux`	Contains source code of demo applications for linux.
`ppc03`	Contains source code of demo applications for linux.

Appendix C. Changelog

C.1. Components

C.1.1. NCore Library

Version 2.4.0.0

- ADD: Integration with Win32 and COM errors on Windows.
- ADD: NStream module.

Version 2.3.1.0

- ADD: NProcessorInfo module for CPU identification on Windows.

Version 2.3.0.1

- FIX: Memory leak in parameters framework.

Version 2.3.0.0

- ADD: HNStream type.
- ADD: Stream integration with .NET.
- UPD: Exception integration with .NET.

Version 2.2.2.0

- CHN: NMemory interface.

Version 2.2.1.0

- ADD: More robust error handling on Windows.

Version 2.2.0.0

- ADD: Unicode support.

Version 2.1.0.2

- FIX: Functions' calling convention on Windows.

Version 2.1.0.1

- UPD: Minor updates.

Version 2.1.0.0

- REM: Registration error codes.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 2.0.1.1

- CHN: Minor changes.

Version 2.0.1.0

- ADD: NParameters module instead of NMetaTypes module for internal infrastructure support.
- CHN: Infrastructure optimization for 64-bit support.

Version 2.0.0.0

- ADD: A lot of stuff for internal infrastructure support.
- CHN: Some changes in internal infrastructure support.

Version 1.0.0.2

- ADD: NIndexPair structure.

Version 1.0.0.1

- FIX: Minor fixes in headers.

Version 1.0.0.0

Initial release.

C.1.2. NImages Library

Version 2.2.0.1

- FIX: Some TIFF files reading.

Version 2.2.0.0

- ADD: I/O with HNStream.
- FIX: Some BMP RLE-compressed files reading.

Version 2.1.0.2

- FIX: Saving in JPEG format for some images.

Version 2.1.0.1

- UPD: Minor updates.

Version 2.1.0.0

- ADD: JPEG format support.

Version 2.0.1.2

- FIX: Minor fixes.

Version 2.0.1.1

- FIX: Reading of some BMP files.

Version 2.0.1.0

- ADD: Unicode support.

Version 2.0.0.5

- FIX: Fixed some TIFF files reading.

Version 2.0.0.4

- FIX: Fixed some bad-formed BMP files reading.

Version 2.0.0.3

Initial release.

C.1.3. FncExtractor Library

Version 2.1.0.0

- UPD: Improved extraction speed.
- ADD: FnceFastExtractStartXxx, FnceFastExtractFinishXxx and FnceFastExtractEnd functions enabling faster extraction for verification/identification with few templates scenarios.

Version 2.0.1.3

- FIX: Memory leak.
- UPD: Improved performance in some modes.

Version 2.0.1.2

- FIX: G computation.

Version 2.0.1.1

- FIX: Generalization with small templates.

Version 2.0.1.0

- UPD: Image quality determination.

Version 2.0.0.0

Initial release. Contains new and more reliable version of features extraction algorithm than in FingerCell 1.2 with a new interface.

C.1.4. FncMatcher Library

Version 2.1.0.0

- UPD: Updated for FncExtractor 2.1.

Version 2.0.0.3

- FIX: Memory leak.

Version 2.0.0.2

- FIX: Matching threshold usage.

Version 2.0.0.1

- FIX: Scores normalization.

Version 2.0.0.0

Initial release. Contains new and more reliable version of features matching algorithm than in FingerCell 1.2 with a new interface.

