VeriFinger 5.0 SDK

VeriFinger 5.0 SDK

Published October 12, 2006. Version 5.0.0.6

Table of Contents

List of Tables

Chapter 1. About

1.1. Introduction

VeriFinger SDK consists of NFRecord, VFExtractor, FPScannerMan, VFMatcher main libraries (Neurotec.Biometrics.NFRecord, Neurotec.Biometrics.VFExtractor, Neurotec.Biometrics.FPScannerMan, Neurotec.Biometrics.VFMatcher in .NET). NCore and NImages (Neurotec, Neurotec.Images in .NET) are auxiliary libraries provides infrastructure and functionality for working with images. The samples programs were developed to use VeriFinger SDK easier.

VeriFinger SDK can be used in a number of compilers under any of these operating systems: Windows 95/98/Me, Windows NT/2000/XP, Linux and Mac OS X.

1.2. Platforms Supported

VeriFinger SDK supports platforms based on x86 processor architecture. Libraries for Windows, Linux and Mac OS X operating systems are provided.

1.3. System Requirements

- PC or Mac with Pentium-compatible 500MHz processor or better.
- Microsoft Windows 9x/ME/NT/2000/XP/2003 or Linux (based on glibc 2.2.5 or compatible) or Mac OS X (10.3.9 or newer).
- Fingerprint scanner driver (users can use the driver included in VeriFinger SDK or can obtain the driver from the scanner's manufacturer).

1.4. Licensing

Customers should obtain a license for VeriFinger SDK only once, and then they will have to order only licenses for VeriFinger installation for each copy of their biometric system.

A license for VeriFinger is required for each running instance of a VeriFinger-based application. The following license types are available:

- Single computer license
- Enterprise license

VeriFinger SDK includes one VeriFinger installation license.

Please also refer to VeriFinger SDK Software License Agreement (\documentation\license.html) for all licensing terms and conditions.

1.4.1. Single computer license

This license type requires VeriFinger installation activation before using the VeriFinger algorithm. Activation can be done using email or through the supplied activation application.

1.4.2. Enterprise license

The enterprise license means unlimited license to use the VeriFinger algorithm in a certain territory or/and for a certain project. These limitations would be included in the licensing agreement. The license price depends on project-specific and appointed limitations.

Chapter 2. What's New

2.1. Version 5.0.0.0

- Improved reliability of fingerprint template extraction and matching.
- Matching algorithm has now one speed and is faster than high speed and more reliable than low speed in VeriFinger 4.2.
- Better quality of input image control.
- Added algorithm optimizations for more fingerprint scanners.
- New SDK structure.
- VFExtractor and VFMatcher libraries should be now used instead of VFinger library to use all features of VeriFinger algorithm.
- FPScannerMan library should be now used instead of ScanMan library to use all features of scanner support.
- .NET wrappers and sample for SDK libraries.

2.2. Version 4.2.1.10

- BiometriKa FX3000 support added and FX 2000 support updated.
- UPEK TCRU2C support added and TCRU1C support updated.
- ADD: NitGen Fingkey Hamster and HamsterII support added.
- ADD: AuthenTec AES2501 support added.
- ADD: Database checking sample added into "Support" folder.

2.3. Version 4.2.1.9

- ADD: Futronic FS80 support added.

2.4. Version 4.2.1.8

- ADD: Testech Bio-I scanner support added.
- CHN: "Install" folder structure changed.

2.5. Version 4.2.1.6

- Fujitsu MBF200 support added.

2.6. Version 4.2.1.5

- Digent Izzix FD1000 scanner support added.

2.7. Version 4.2.1.3

- Atmel, Tacoma CMOS, STARTEK FM200, LighTuning scanners support added.

2.8. Version 4.2.1.2

- Visual C++ MFC sample added.

2.9. Version 4.2.1.0

- Updated Atmel scanner mode.

2.10. Version 4.2.0.0

- Improved reliability.
- Better matching performance. Both matching speeds are now about 50% faster than in version 4.1.
- Reduced template size. Now template occupies 150 - 300 bytes (vs. 200 - 650 in version 4.1).
- Features compression and decompression functions are now built in the VFinger library.
- Added CrossMatch Verifier 300 scanner mode.
- VeriFinger can now return skeletonized image.
- Added separate "What's new" sections for VeriFinger library, VeriFinger SDK and VeriFinger SDK documentation.

2.11. Version 4.1.0.0

- Completely new interface of VFinger library.
- Higher matching speed. Now only two speeds are available - low (0 speed in 4.0) and high (5 speed in 4.0). Both speeds are faster that in version 4.0.
- Improved recognition reliability. Both speeds are more reliable that in version 4.0.
- Optimizations for fingerprint scanners. Optimizations for a number of scanners are available.
- ScanMan library as a substitution of UrUReader, ASReader and FX2KReader in VeriFinger SDK 4.0. Now you can work with any scanner it supports via the same interface.

Chapter 3. Overview

3.1. Fingerprint Scanner Support

Fingerprint scanner support allows live scanning of fingerprint for further features extraction Neurotec.Biometrics.VFExtractor. Scanner support is implemented in FPScannerMan or NFRecord class in .NET and in a number of libraries on Linux (Scanners API for Linux OS).

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

Table 3.1. Supported scanners for different platforms

Scanner	MS Windows	Linux	Mac OS X
Atmel FingerChip	x		
Authentec AES2501B	x		
Authentec AES4000	x	x	x
Authentec AF-S2	x	x	x
BiometriKa FX 2000	x	x	
BiometriKa FX 3000	x	x	
Cross Match Verifier 300-USB	x	x	
Green Bit DactyScan 26	x		
Digent Izzix	x		
DigitalPersona U.are.U	x		
Ethentica Ethenticator USB 2500	x		
Fujitsu MBF200	x	x	x
Futronic FS80	x		
Identix DFR2080	x		
Identix DFR2090	x		

Scanner	MS Windows	Linux	Mac OS X
Identix DFR2100	x		
Identix TouchView	x		
LighTuning LTT-C500	x		
NITGEN Fingkey Hamster	x		
Secugen Hamster	x		
Shimizu/Tacoma CMOS	x	x	x
Startek FM200	x	x	x
Testech Bio-i	x		
UPEK TouchChip	x		

3.1.1. Scanners API for Linux OS

This file has `scanner_info` structure:

```
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;
};
extern struct scanner_info scanner_fx2k;
extern struct scanner_info scanner_shimizu;
extern struct scanner_info scanner_mbf200;
extern struct scanner_info scanner_fm200;
extern struct scanner_info scanner_cross_match;
```

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. Returned memory can't be released, and is overwritten with second scan.
int timeout;	User adjustable timeout for USB requests.
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 calculated 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.

Important

These drivers are not thread safe.

Example:

```
#include <stdlib.h>
#include <scanner.h>
struct scanner_info* scanner = &scanner_cross_match;
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 scanner/cross_match_drv.o -o sample -lusb
```

`cross_match_drv.o` requires `libusb` and must be linked with `-lusb`

`fm200_drv.o` requires `libusb` and must be linked with `-lusb`

`mbf200_drv.o` requires `libusb` and must be linked with `-lusb`

`shimizu_drv.o` requires `libusb` and must be linked with `-lusb`

`fx2k.o` must be linked with `-lfx3L -lfx3scan -lfxoem -lm`

3.2. Fingerprint View Control

The fingerprint view control NFView (only for .NET) is used to show a fingerprint image and extracted minutia points.

3.3. Image Support

Image support in the VeriFinger SDK 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. Provides a way to learn which image formats are supported and to load and

save images in 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.

3.3.1. 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 VeriFinger SDK is defined by HNImage handle in NImage module (`NImage` class in .NET). It is an encapsulation of a memory block containing image pixels and is organized as rows following each other in top-to-bottom order (number of rows is equal to image height). Each row occupies stride bytes and is organized as pixels following each other in left-to-right order (number of pixels in the row is equal to image width). Each pixel is described by pixel format. See `NImageGetWidth`, `NImageGetHeight`, `NImageGetStride`, `NImageGetPixelFormat` and `NImageGetPixels` functions (Width, Height, Stride, PixelFormat and Pixels properties in .NET) for more information.

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

An image can be created either new or from existing memory block. See `NImageCreate`, `NImageCreateFromData` and `NImageCreateWrapper` functions (`Create`, `FromData` and `GetWrapper` methods in .NET) for more information.

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

An image can be converted to different pixel format using `NImageCreateFromImage` or `NImageCreateFromImageEx` function (`FromImage` method in .NET).

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

- On Windows `BmpSaveImageToHBitmap` function (`ToHBitmap` method in .NET) can be used to receive a standard Win32 HBITMAP for the image. The reverse process is also possible using `BmpLoadImageFromHBitmap` function (`FromHBitmap` method in .NET).
- In .NET `ToBitmap` method can be used to receive a standard .NET Bitmap. The reverse process is also possible using `FromBitmap` method.
- On Linux there is no easy method implemented. However, image provides access to

memory block containing its pixels via `NImageGetPixelFormat` function (PixelFormat method in .NET). 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 (`Save` method in .NET).

An image stored in file in any supported image format can be loaded using `NImageCreateFromFile` function (`FromFile` method in .NET).

3.3.2. 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 VeriFinger SDK is defined by HNImageFormat handle in NImageFormat module (`NImageFormat` class in .NET).

There is a number of image formats supported in the VeriFinger SDK. Not for all of them both reading and writing from/to file is supported on all platforms. See the following table for details.

Image Format	**Can read**	**Can write**
BMP	Yes	Yes
GIF	In .NET only	In .NET only
JPEG	In .NET only	In .NET only
PNG	In .NET only	In .NET only
TIFF	Yes	In .NET only

These image formats are accessible using `NImageFormatGetBmp` and `NImageFormatGetTiff` functions (*`Bmp`*, *`Gif`*, *`Jpeg`*, *`Png`* and *`Tiff`* read-only fields in .NET).

To learn which image formats are supported in the VeriFinger SDK in version-independent way `NImageFormatGetFormatCount` and `NImageFormatGetFormat` functions should be used (Formats property in .NET).

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

To learn which image format should be used to read or write a particular file `NImageFormatSelect` function (`Select` method in .NET) should be used.

An image can be loaded and saved from/to file or memory buffer using `NImageFormatLoadImageFromFile`, `NImageFormatLoadImageFromMemory`, `NImageFormatSaveImageToFile` and `NImageFormatSaveImageToMemory` functions (`LoadImage` and `SaveImage` methods in .NET). Note that not all image formats supports both reading and writing. Use `NImageFormatCanRead` and/or `NImageFormatCanWrite` function(s) (CanRead and/or CanWrite property(ies) in .NET) to check if the particular image format does.

3.3.3. Low-Level Image Input-Output

Low-level image I/O in the VeriFinger SDK is implemented in Bmp and Tiff modules (`Bmp` and `Tiff` classes in .NET).

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

Those functions (static methods in .NET) can take parameters that precisely control loading and saving of the image in particular formats.

Chapter 4. Using

4.1. VeriFinger SDK

4.1.1. Windows

Before using VeriFinger SDK, please execute command:

```
id_gen.exe serial_number_file
```

Example:

```
id_gen.exe sn.txt
```

Send the created "verifinger.id" file to Neurotechnologija (sales@neurotechnologija.com) or distributor from which library was acquired. Then save received VeriFinger SDK id file in the same as "pg.exe" directory.

Please run "register.bat", that creates and runs Neurotechnologija service. Make sure that Neurotechnologija service is started every time while using VeriFinger SDK and `pgd.conf` shows the license file place ("LicenceFile = neurotechnologija.lic" shows that license file `neurotechnologija.lic` is in system directory (i.e. "C:\WINDOWS\neurotechnologija.lic"), otherwise it is necessary to write full path to the license file.

4.1.2. Linux and Mac OS X

Before using VeriFinger SDK, please execute command:

```
./id_gen serial_number_file
```

Example:

```
./id_gen sn.txt
```

Send the created "`verifinger.id`" file to Neurotechnologija (sales@neurotechnologija.com) or distributor from which library was acquired. Then save received VeriFinger SDK id file in the same as "pgd" directory.

Please run "./pgd". Make sure that it is running while using VeriFinger SDK and `pgd.conf` shows the right license file place ("LicenceFile = neurotechnologija.lic" shows that license file `neurotechnologija.lic` is in current directory.

4.2. VeriFinger SDK Trial

4.2.1. Windows

Before using VeriFinger SDK, please run "register.bat" from "\bin\Win32_x86\" folder.

4.2.2. Linux and Mac OS X

Before using VeriFinger SDK, please run "./pgd" from "bin/" folder.

Chapter 5. Samples

Before running samples read Using chapter.

5.1. Windows

5.1.1. Microsoft Visual C#

Source of Microsoft Visual C# applications is located in `samples\dotNET\` directory of SDK for Visual Studio 2005.

5.1.2. Microsoft Visual C++

Source of Microsoft Visual C++ (later referenced as C) application is located in `samples\Windows\VFDemo.cpp` directory of SDK. Project file for C is `VFDemo.cpp.vcproj` (version 7.1).

Interfaces are provided in `Include/` directory.

Demo application does not use any standard windowing libraries (like MFC), so it can be easily ported to compilers other than Microsoft Visual C++. Object-oriented interface to Win32 API is implemented in core files instead:

`System.h`	Precompiled header file for the project includes standard Windows headers, standard C and standard C++ libraries headers, `Strings.h`, `Vectors.h`, `Exceptions.h` and `System.rh` headers and defines several macros.
`System.cpp`	Uses `System.h` header for precompiled header files generation and defines empty string constant (`empty_str`).
`Strings.h`	Defines string class that encapsulates C string of TCHAR's (can be ANSI or Unicode character set), functions that work with C strings and functions to perform ANSI C string/string class conversion.
`Vectors.h`	Defines vector template class that can be used to make vectors (collections) of objects of any type. Also defines vector of strings (strings).
`Exceptions.h`	Defines exceptions, uses standard C++ library exception header. Also defines `EXCEPTION` inline function to create exception class object from TCHAR string.

`Utils.h` `Utils.cpp`	Various utility functions.
`Controls.h` `Controls.cpp`	Define `CControl`, `CForm` and `CApplication` classes and a number of user interface functions. `CControl` is a base class for Windows controls. `CForm` is a base class for application main window or dialog (that contain controls). `CApplication` is a base class for Windows GUI application.
`Dialogs.h` `Dialogs.cpp`	Define classes for standard Windows dialogs like `COpenFileDialog` for standard file open dialog.
`StdControls.h` `StdControls.cpp`	Define classes for standard Windows controls like `CButton` for the button control, `CEdit` for the edit control, etc.
`Threads.h` `Threads.cpp`	Define `CThread` class that encapsulates Win32 thread.
`Registry.h` `Registry.cpp`	Define `CRegistry` class that encapsulates Windows registry.
`Log.h` `Log.cpp`	Define `CLog` class that can be used for logging information.

These files are not intended for use in retail applications, they are not heavily tested. For retail application you should use standard libraries (MFC, Qt, WTL, WxWidgets).

Application defines class `CVFImage` for fingerprint image encapsulation. Images can be loaded from file and saved via methods of this class. Internally they use image input/output class (`CVFImageIO`) that uses number of registered image formats (`CVFImageFormat`) to perform these operations. These classes are defined in `VFImage.h` and `VFImage.cpp` files. Descendants of image format class for bitmap and TIFF file formats are defined: `CVFImageFormatBMP` and `CVFImageFormatTIFF` (`VFImageFormatBMP.h`, `VFImageFormatBMP.cpp`, `VFImageFormatTIFF.h`, `VFImageFormatTIFF.cpp`) classes.

Also application uses a class `CVFFeatures` for fingerprint features encapsulation (`VFFeatures.h`, `VFFeatures.cpp`).

To display fingerprint image and features a control is defined: `CVFView` class is defined (`VFControls.h`, `VFControls.cpp`).

To store, and maintain fingerprint features in and load from ADO database `CVFDatabase` class is defined (`VFDatabase.h`, `VFDatabase.cpp`).

Functions that work with application options are defined in `VFOptions.h` and `VFOptions.cpp` (application only options - in `Options.h` and `Options.cpp` files) files.

Work with file lists and finger identifiers is organized in `VFFileList.h` and `VFFileList.cpp` files.

`AboutForm.h` and `AboutForm.cpp` files implement About dialog (`CAboutForm`). This dialog shows information about sample application.

`VFOptionsForm.h` and `VFOptionsForm.cpp` (application only options - in `OptionsForm.h` and `OptionsForm.cpp` files) files implement application options editing dialog (`COptionsForm`).

`FileListEntryForm.h` and `FileListEntryForm.cpp` files implement File list entry form (`CFileListEntryForm`).

`FileListForm.h` and `FileListForm.cpp` files implement File list editor form (`CFileListForm`).

`WaitForm.h` and `WaitForm.cpp` files implement Wait dialog (`CWaitForm`). It is shown when user is waiting for completion of computationally expensive operation.

`MainForm.h` and `MainForm.cpp` files implement main application form (`CMainForm`). Application logic is implemented here.

In `CVFDemoApp` class as a descendant of `CConsoleApplication` is defined and created in `WinMain` function. Both create main form and run it.

Main form in the constructor creates controls (calls `InitControls` function), initializes internal variables. Controls are destroyed automatically. Internal variables are uninitialized in main form destructor.

Then main and scanner logs are initialized and information about application, operating system components are displayed there. Then options and component registration information is loaded and updated. Then application state is updated.

When main form window is created (`OnCreate` method) application state that requires window to be created is updated (that is linked to menu items), scanner monitoring is started and available scanners are enumerated and capturing from those scanners is started. When main form window is destroyed (`OnDestroy`) capturing from all scanners is stopped and scanner monitoring is stopped.

When form is resized (`OnResize`) or splitters are moved, controls are rearranged.

When form is shown on the screen for the first time (`OnFirstShow`) registration dialog is displayed if components is not registered and information about library registration is displayed in the log. Then database loading is started (`StartDBOpen`). When the form is closed (`OnClose`) database is closed using a database close thread (`CDBCloseThread`) and progress dialog is displayed until it finishes.

`StartDBOpen` searches for a database file (`VFDemo.mdb`) in current directory and creates default database (`CreateDefaultDatabase`) if the file does not exist. Then starts a database loading thread (`CDBOpenThread`) via `StartTask` method.

`StartTask` method changes mode to `vfdmNone` (if task is not file list) and starts specified task.

When task thread completes `OnTaskComplete` method is called which changes mode to one that was before task started. Before it calls appropriate task completion routine (`OnDBOpenComplete`, `OnFileListComplete`, etc.).

Then main form responds to user command selection from menu and controls through `OnCommand` method:

C `WM_COMMAND` identifier and control (or menu) in OnCommand method, main form method	Source, description
`ID_FILE_CLEAR` `Clear`	File»Clear command. Clears the output of the application
`ID_FILE_OPEN` `OnFileOpen`	File»Open command. Shows file open dialog where user selects image files to open, then makes file list from selected files and calls `StartFileList` method
`ID_FILE_OPENFILELIST` `OnFileOpenFileList`	File»Open file list... command. Shows file open dialog where user selects file list file then loads the file list and calls `StartFileList` method
`ID_FILE_SAVE` `OnFileSave`	File»Save... command. Saves image in the left window using `SaveImage` method

`ID_FILE_SAVERIGHT` `OnFileSaveRight`	File»Save right... command. Saves in the right window using `SaveImage` method
`ID_FILE_STOP` `OnFileStop`	File»Stop... command. Stops current task using `StopTask` method
button_main_stop event `BN_CLICKED`	Stop button in Main log panel. Stops current task using `StopTask` method
button_results_stop event `BN_CLICKED`	Stop button on Identification results panel. Stops identification using `StopIdentification` method
`ID_FILE_EXIT` `OnFileExit`	File»Exit command. Closes main form
`OnCloseQuery`	Query for main form close. Prompts to stop current task
`OnClose`	Form is closing. Closes database in database close thread (`CDBCloseThread`) and displays wait dialog until thread finishes
`ID_MODE_ENROLLMENT` `OnModeEnrollment`	Mode»Enrollment command. Changes mode to `vfdmEnrollment` (via `SetMode` method)
`ID_MODE_ENROLLMENTWITHGEN` `OnModeEnrollmentWithGen`	Mode»Enrollment With Generalization command. Changes mode to `vfdmEnrollmentWithGen` (via `SetMode` method)
`ID_MODE_VERIFICATION` `OnModeVerification`	Mode»Verification command. Changes mode to `vfdmVerification` (via `SetMode` method)
`ID_MODE_IDENTIFICATION`	Mode»Identification command. Changes mode to `vfdmIdentification` (via

`OnModeIdentification`	`SetMode` method)
`ID_VIEW_ZOOMIN` `OnViewZoomIn`	View»Zoom in command. Zooms in in left and right views
`ID_VIEW_ZOOMOUT` `OnViewZoomOut`	View»Zoom out command. Zooms out in left and right views
`ID_VIEW_ZOOM1TO1` `OnViewZoom1To1`	View»Zoom 1:1 command. Zooms to 100% in left and right views
`ID_VIEW_SHOWFEATURES` `OnViewShowFeatures`	View»Show features command. Toggles fingerprint features on the right view
`ID_VIEW_SHOWSINGULARPOINTS` `OnViewShowSP`	View»Show singular points command. Toggles singular points on the right view
`ID_VIEW_SHOWBLOCKEDORIENTATIONS` `OnViewShowBO`	View»Show blocked orientations command. Toggles blocked orientations on the right view
`ID_TOOLS_FILELISTEDITOR` `OnToolsFileListEditor`	Tools»File list editor... command. Opens File list editor window
`ID_TOOLS_CLEARDATABASE` `OnToolsClearDatabase`	Tools»Clear database command. Clears database
`ID_TOOLS_CLEARMAINLOG` `OnToolsClearMainLog`	Tools»Clear main log command. Clears main log
`ID_TOOLS_CLEARSCANNERLOG` `OnToolsClearScannerLog`	Tools»Clear scanner log command. Clears scanner log

`ID_TOOLS_OPTIONS` `OnToolsOptions`	Tools»Options... command. Shows Options dialog where user edits application options
`ID_TOOLS_ENGINEOPTIONS` `OnToolsEngineOptions`	Tools»VeriFinger Options... command. Shows VeriFinger Options dialog where user edits VeriFinger options
`ID_HELP_ABOUT` `OnHelpAbout`	Help»About... command. Shows About dialog

Application loads images from files and receives them from scanners. `StartFileList` method is called when user opens files or file list. In there is only one file in file list then calls `OnFileImage` method directly. In other case starts file list thread (`CFileListThread`) via `StartTask` method. During execution of the thread `OnFileImage` method is called for each file in file list, which suspends the thread and calls `OnFileImage` method. When thread finishes `OnFileListComplete` method is called from `OnTaskComplete` method. `OnFileImage` method loads image from file and calls `OnImage` method. When image is received from a scanner `OnScannerImage` method is called, which calls `OnImage` method. When capturing state is changed for a scanner `OnScannerState` method is called, which displays status information in scanner log. `OnSMMonitor` method is called when scanner event occurs; displays event information in scanner log.

`OnImage` method starts features extraction from the image in a thread (`CExtractionThread`). Thread uses VFExtract function. When thread finishes `OnExtractionComplete` method is called. This method regarding to current mode calls on of these methods:

`Enroll` - for `vfdmEnrollment` - stores extracted features in a database.

`EnrollWithGen` - for `vfdmEnrollmentWithGen` - performs features generalization for `VF_GENERALIZE_COUNT` extracted features collections (uses VFGeneralize function) then calls `Enroll` method.

`Verify` - for `vfdmVerification` - performs verification on two extracted features collections (uses VFVerify function).

`StartIntentification` - for `vfdmIdentification` - starts identification thread (`CIdentificationThread`), which performs identification of extracted features in the database (uses VFIdentifyStart, VFIdentifyNext, VFIdentifyEnd functions). When thread finishes `OnIdentificationComplete` method is called, which displays identification information in the log.

5.1.3. Microsoft Visual C++ (MFC)

Source of Microsoft Visual C++ (MFC) (later referenced as MFC) application is located in `samples\Windows\VFDemo.MFC` directory of SDK. Project file for MFC is `VFDemo.MFC.vcproj`.

Interfaces for components are provided in `Include/` directory of SDK. Application also uses DLL library files in `Lib/` directory of SDK. Application uses standard libraries for Windows (MFC)

Application defines class for fingerprint image encapsulation. It is `CVFImage`. This class is defined in `VFImage.h` and `VFImage.cpp` files. Also application uses a class for fingerprint features encapsulation: `CVFFeatures` (`CVFFeatures.h` and `CVFFeatures.cpp`)

To store, and maintain fingerprint features in and load from ADO database `VFDDatabase` class is defined (`VFDDatabase.h`, `VFDDatabase.cpp`).

Functions that work with application options are defined in `VFDemoOptions.h` and `VFDemoOptions.cpp` files. Some functions are in `VFDemoDlg.cpp` file.

Main dialog form in the constructor initialize internal variables and in `CVFDemoDlg::OnInitDialog()` function creates menu. All controls are created automatically (drawn using resource view in Microsoft Visual C++ project).

When "Start capturing" button pressed scanner monitoring is started, available scanners are enumerated, capturing from those scanners is started and fingerprint are to be loaded from database to memory. "Start capturing" button caption changes to "Stop capturing". When "Stop capturing" button is pressed or main dialog form exits, capturing from all scanners is stopped and scanner monitoring is stopped. Also database connection is closed.

Application receives images from scanners. During fingerprint scanning `OnScannerState` method displays status information in scanner log. After image was given from scanner, user can press "Extract" button to extract features from image. After pressing this button `VFExtract` function is called. During extraction user must enter finger ID. If not valid or empty fingerprint ID entered, it will not be added to database. After this action, processed image appears on screen (right view).

Application can search for duplicates (if this enabled in options). Searching for duplicates and identifying application uses same `VFIdentifyStart`, `VFIdentifyNext`, `VFIdentifyEnd` functions. All information about identification results (identifies processed fingerprint with fingerprints from database) is shown in report list field called "Identification results".

Working with application all main events is displaying in main and scanner logs such us image info, database events, scanner states, etc. Also user can save original image and processed image too.

5.1.4. Borland Delphi 6

Source of application is located in `samples\Windows\VFDemo.pas` directory. Project file - `VFDemo_pas.dpr`.

Interface for components are provided in `Include/Windows/` directory of SDK: `VFinger.pas` modules.

Delphi application use standard VCL library and few extension modules:

`Utils.pas`	Various utility functions.
`Log.pas`	`TLog` class. Used for information logging.

Applications define class `TVFImage` for fingerprint image encapsulation. Image can be loaded from file and saved via methods of this class. Internally they use image input/output class (`TVFImageIO`) that uses number of registered image formats (`TVFImageFormat`) to perform these operations. These classes are defined in `VFImage.pas` module. Descendants of image format class for bitmap and TIFF file formats are defined: `TVFImageFormatBMP` and `TVFImageFormatTIFF` (`VFImageFormatBMP.pas`, `VFImageFormatTIFF.pas`).

Also application uses a class `TVFFeatures` for fingerprint features encapsulation (`VFFeatures.pas`).

To display fingerprint image and features a control is defined: `TVFView` class is defined (`VFControls.pas`).

To store, and maintain fingerprint features in and load from ADO database `TVFDatabase` class is defined (`VFDatabase.pas`).

Functions that work with application options are defined in `VFOptions.pas` module.

Work with file lists and finger identifiers is organized in `VFFileList.pas` module.

`FormAbout.pas` and `FormAbout.dfm` modules implement About dialog (`TAboutForm`). This dialog shows information about sample application.

`FormOptions.pas` and `FormOptions.dfm` modules in implement application options editing dialog (`TOptionsForm`).

`FormFileListEntry.pas` `FormFileListEntry.dfm` modules implement File list entry form (`TFileListEntryForm`).

`FormFileList.pas` `FormFileList.dfm` modules implement File list editor form (`TFileListForm`).

`FormWait.pas` and `FormWait.dfm` modules implement Wait dialog (`TWaitForm`). It is shown when user is prompted to wait until a long operation is complete.

`FormMain.pas` and `FormMain.dfm` modules implement main form (`TMainForm`) -

main application window user works with. Main application logic is implemented here.

Main project file - `VFDemo_pas.dpr`.

Internal variables are initialized in `FormCreate`. Controls are destroyed automatically. Internal variables are uninitialized in `FormDestroy` method.

Then main and scanner logs are initialized and information about application, operating system and components are displayed there. Then options components registration information is loaded and updated. Then application state is updated.

When main form window is created (`FormCreate`) application state that requires window to be created is updated (that is linked to menu items), scanner monitoring is started and available scanners are enumerated and capturing from those scanners is started. When main form window is destroyed (`FormDestroy`) capturing from all scanners is stopped and scanner monitoring is stopped.

When form is resized (`FormResize`) or splitters are moved, controls are rearranged.

When form is shown on the screen for the first time (`OnFirstShow`) registration information about library registration is displayed in the log. Then database loading is started (`StartDBOpen`). When form is closed (`FormClose`) database is closed in a database close thread (`TDBCloseThread`) and wait dialog is displayed until it finishes.

`StartDBOpen` searches for a database file with name `VFDemo.mdb` in current directory and if not found creates default database (`CreateDefaultDatabase`). Then starts a database loading thread (`TDBOpenThread`) via `StartTask` method.

`StartTask` method changes mode to vfdmNone (if task is not file list) and starts specified task

When task thread completes `OnTaskComplete` method is called which changes mode to one that was before task started. Before it calls appropriate task complete routine (`OnDBOpenComplete`, `OnFileListComplete`, etc.).

Then main form responds to user command selection from menu and controls through actions' `OnExecute` events:

Delphi action's `OnExecute` event or form event (as main form method)	Source, description
`FileClearExecute`	File»Clear command. Clears the output of the application
`FileOpenExecute`	File»Open command. Shows file open dialog where user selects image files to open, then makes file list from selected files and calls `StartFileList` method

`FileOpenFileListExecute`	File»Open file list... command. Shows file open dialog where user selects file list file then loads the file list and calls `StartFileList` method
`FileSaveExecute`	File»Save... command. Saves left image using `SaveImage` method
`FileSaveRightExecute`	File»Save right... command. Saves right image using `SaveImage` method
`FileStopExecute`	File»Stop... command. Stops current task using `StopTask` method
`ButtonMainStopClick`	Stop button in Main log panel. Stops current task using `StopTask` method
`ButtonResultsStopClick`	Stop button on Identification results panel. Stops identification using `StopIdentification` method
`FileExitExecute`	File»Exit command. Closes main form
`FormCloseQuery`	Query for main form close. Prompts to stop current task
`FormClose`	Form is closing. Closes database in database close thread (`TDBCloseThread`) and displays wait dialog until thread finished
`ModeEnrollmentExecute`	Mode»Enrollment command. Changes mode to `vfdmEnrollment` (via `SetMode` method)
`ModeEnrollmentWithGenExecute`	Mode»Enrollment With Generalization command. Changes mode to `vfdmEnrollmentWithGen` (via `SetMode` method)

`ModeVerificationExecute`	Mode»Verification command. Changes mode to `vfdmVerification` (via `SetMode` method)
`ModeIndetificationExecute`	Mode»Identification command. Changes mode to `vfdmIdentification` (via `SetMode` method)
`ViewZoomInExecute`	View»Zoom in command. Zooms in in left and right views
`ViewZoomOutExecute`	View»Zoom out command. Zooms out in left and right views
`ViewZoom1To1Execute`	View»Zoom 1:1 command. Zooms to 100% in left and right views
`ViewShowFeaturesExecute`	View»Show features command. Toggles show or not features on right view
`ViewShowSingularPointsExecute`	View»Show singular points command. Toggles singular points on the right view
`ViewShowBlockedOrientationsExecute`	View»Show blocked orientations command. Toggles blocked orientations on the right view
`ToolsFileListEditorExecute`	Tools»File list editor... command. Opens File list editor window
`ToolsClearDatabaseExecute`	Tools»Clear database command. Clears database
`ToolsClearMainLogExecute`	Tools»Clear main log command. Clears main log
`ToolsClearScannerLogExecute`	Tools»Clear scanner log command. Clears scanner log
`ToolsOptionsExecute`	Tools»Options... command. Shows Options

	dialog where user edits application options
	Tools»VeriFinger Options... command. Shows VeriFinger Options dialog where user edits VeriFinger options
`HelpAboutExecute`	Help»About... command. Shows About dialog

Application loads images from files and receives them from scanners. `StartFileList` method is called when user opens files or file list. In there is only one file in file list then calls `OnFileImage` method directly. In other case starts file list thread (`TFileListThread`) via `StartTask` method. During execution of the thread `OnFileImage` method is called for each file in file list, which suspends the thread and calls `OnFileImage` method. When thread finishes `OnFileListComplete` method is called from `OnTaskComplete` method. `OnFileImage` method loads image from file and calls `OnImage` method. When image is received from a scanner `OnScannerImage` method is called, which calls `OnImage` method. When capturing state is changed for a scanner `OnScannerState` method is called, which displays status information in scanner log. `OnSMMonitor` method is called when scanner event occurs; displays event information in scanner log.

`OnImage` method starts features extraction from the image in a thread (`TExtractionThread`). Thread uses VFExtract function. When thread finishes `OnExtractionComplete` method is called. This method regarding to current mode calls on of these methods:

`Enroll` - for `vfdmEnrollment` - stores extracted features in a database.

`EnrollWithGen` - for `vfdmEnrollmentWithGen` - performs features generalization for `VF_GENERALIZE_COUNT` extracted features collections (uses VFGeneralize function) then calls `Enroll` method.

`Verify` - for `vfdmVerification` - performs verification on two extracted features collections (uses VFVerify function).

`StartIntentification` - for `vfdmIdentification` - starts identification thread (`TIdentificationThread`), which performs identification of extracted features in the database (uses VFIdentifyStart, VFIdentifyNext, VFIdentifyEnd functions). When thread finishes `OnIdentificationComplete` method is called, which displays identification information in the log.

5.1.5. Microsoft Visual Basic 6.0

Source of Visual Basic 6.0 application is located in `samples\Windows\VFDemo.bas\` subdirectory of SDK. Files which are located in Visual Basic 6.0 sample directory are listed and described here:

Visual Basic 6.0 Forms	
File	Description
`MainForm.frm`	Main form of sample application. Contains main menu, two picture boxes (for original and binary fingerprint image), log window and progress bar
`DialogSettings.frm`	Application settings form. Allows change similarity threshold by changing FAR (false acceptance rate) and image resolution (DPI)
`DialogSettings.frx`	Settings form data (binary file; used by Visual Basic)
`ScannerList.frm`	Scanners list form. Displays list of scanners and allow choose one
`ScannerList.frx`	Scanners list form data (binary file; used by Visual Basic)
`DialogRegister.frm`	VeriFinger DLL registration form
Visual Basic 6.0 Modules	
File	Description
`VFinger.bas`	VeriFinger interface declaration and implementation of helper functions
`Service.bas`	Useful collections of functions/sub for fingerprint features drawing and image manipulations
`DataBase.bas`	Operations with database
`Bitmaps.bas`	Operations with bitmaps (currently only save to file)
Visual Basic 6.0 Class Modules	

File	Description
`FingerprintRecord.cls`	Class encapsulating fingerprint record
DLLs	
File	Description
`VFinger.dll`	VeriFinger DLL
`VFVBP42.dll`	VeriFinger Parser DLL
Database	
File	Description
`VFDemo.mdb`	Microsoft Access database
Project	
File	Description
`VFingerVBProject.vbp`	Visual Basic Project
`VFingerVBProject.vbw`	Visual Basic Project Work Space

It is impossible to use VeriFinger DLL directly from Visual Basic 6.0 therefore special parser was written. VeriFinger Visual Basic Parser DLL source is located in `samples\Windows\Wrappers\Visual Basic Parser\` directory.

Visual Basic 6.0 sample application has five possible working modes (user must choose mode when start application):

Mode	Description
Enrollment	Fingerprint enrollment mode. Scanned or loaded fingerprint image

	will be enrolled in to database.
Enrollment with features generalization	Fingerprint enrollment mode. In this mode user must scan or load tree 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.
Recognition (fast)	Fingerprint identification mode. Program will start identification process if you will scan finger or load fingerprint image. In this mode database records are sorted by G and program search until found first match.
Recognition (full)	Fingerprint identification mode. Program will start identification process if user will scan finger or load fingerprint image. In this mode all database records are reviewed and all matches are displayed.
Verification	Two scanned or loaded fingerprint images will be compared in this mode.

Demo application will call VeriFinger DLL functions corresponding to selected mode.

5.1.5.1. VeriFinger Visual Basic Parser

Parser files are listed here:

File	Description
`vf42parser.sln`	Project solution file.
`vf42parser.dsp`, `vf42parser.vcproj`	Project files.
`vf42parser.dsw`	Project workspace.
`VFVBP42.def`	Defines DLL exported functions.

`vf42parser.cpp`	DLL functions implementation.
`vfvbp42.rc`	DLL resources.
`resource.h`	Defines resources identificators.
`2dbytearray.cpp`	Implements class for 2-D arrays.
`2dbytearray.h`	Declares class for 2-D arrays.
`Image.cpp`	Implements image manipulation class.
`Image.h`	Declares image manipulation class.
`VFinger.h`	VeriFinger DLL header
`VFinger.lib`	VeriFinger DLL library

Parser allow for Visual Basic applications pass arrays or structures instead of pointers. There is a lot of code in parser source that manipulates with SAFEARRAY data type. Please refer to MSDN (Microsoft Developer Network) library http://msdn.microsoft.com/ for more information about SAFEARRAYs. Parser also contains fingerprint features decompression/compression functions, which converts array of features to features structure. Functions prefix `"VF"` was changed into `"VBVF"` to prevent name collision with VeriFinger DLL functions.

Exported functions:

```
// Features decompression/compression
INT WINAPI VBVFDecompressFeatures(VARIANT* vfeatures, VFFEATURES* f);
INT WINAPI VBVFCompressFeatures
(
VFFEATURES* f,
VARIANT* vfeatures,
LONG* size
);

// Initialization
INT WINAPI VBVFInitialize();
INT WINAPI VBVFFinalize();

// Registration
INT WINAPI VBVFRegistrationType();
```

```
// Contexts
INT WINAPI VBVFCreateContext();
INT WINAPI VBVFFreeContext(INT context);

// Parameters
INT WINAPI VBVFGetParameterType(LONG parameter)
INT WINAPI VBVFGetParameter(INT parameter, VARIANT* value, INT context)
INT WINAPI VBVFSetParameter(INT parameter, VARIANT value, INT context)

// Features extraction
INT WINAPI VBVFExtract
(
INT width,
INT height,
VARIANT* vimage,
LONG resolution,
VARIANT* vfeatures,
INT* size,
INT context
);

// Features generalization
INT WINAPI VBVFGeneralize
(
INT count,
VARIANT* vgen_features,
VARIANT* vfeatures,
INT* size,
INT context
);

// Verification
INT WINAPI VBVFVerify
(
VARIANT* vfeatures1,
VARIANT* vfeatures2,
void* md,
INT context
);

// Identification
INT WINAPI VBVFIdentifyStart(VARIANT* vtest_features, INT context);
INT WINAPI VBVFIdentifyNext
(
VARIANT* vsample_features,
void* md,
INT context
);

INT WINAPI VBVFIdentifyEnd(INT context);

// Helper functions
INT WINAPI VBImageToHandle
(
INT width,
INT height,
VARIANT* vimage,
INT* handle,
INT Pallete
);

INT WINAPI VBHandleToImage
(
INT handle,
INT* width,
INT* height,
VARIANT* vimage
);

INT WINAPI VBLoadImageFromFile
(
CHAR* filename,
INT* width,
INT* height,
VARIANT* vimage
);

INT WINAPI VBDrawFeatures
(
VFFEATURES* f,
INT width,
INT height,
VARIANT* vimage,
INT pallete,
INT parameters,
INT* handle
);
```

5.1.5.2. Usage of VeriFinger Visual Basic Parser

VeriFinger Parser DLL exports following functions, wrapping VeriFinger functions:

Function	Description
`VBVFInitialize`	For more information please see VfeCreate, VfmCreate
`VBVFFinalize`	For more information please see VfeFree, VfmFree
`VBVFRegistrationType`	For more information please see VfeIsRegistered, VfmIsRegistered
`VBVFCreateContext`	For more information please see VfeCreate, VfmCreate

`VBVFFreeContext`	For more information please see VfeFree, VfmFree
`VBVFGetParameterType`	
`VBVFGetParameter`	For more information please see VFGetParameter
`VBVFSetParameter`	For more information please see VFSetParameter
`VBVFExtract`	For more information please see VFExtract
`VBVFGeneralize`	For more information please see VFGeneralize
`VBVFVerify`	For more information please see VFVerify
`VBVFIdentifyStart`	For more information please see VFIdentifyStart
`VBVFIdentifyNext`	For more information please see VFIdentifyNext
`VBVFIdentifyEnd`	For more information please see VFIdentifyEnd

Additional functions, which help to work with images and fingerprint features:

VFDecompressFeatures	Converts features array to features structure.

Visual Basic:

```
Public Declare Function VFDecompressFeatures L
ib "VFVBP42.dll" Alias "VBVFDecompressFeatures"
(
features As Variant,
structure As VFFEATURES
) As Long
```

Visual Basic .Net:

```
Public Declare Function VFDecompressFeatures
Lib "VFVBP42.dll" Alias "VBVFDecompressFeatures"
(
features As Object,
structure As VFFEATURES
) As Integer
```

VFCompressFeatures

Converts features structure to features array.

Visual Basic:

```
Public Declare Function VFCompressFeatures
Lib "VFVBP42.dll" Alias "VBVFCompressFeatures"
(
structure As VFFEATURES,
features As Variant,
ByRef size As Long
) As Long
```

Visual Basic .Net:

```
Public Declare Function VFCompressFeatures
Lib "VFVBP42.dll" Alias "VBVFCompressFeatures"
(
structure As VFFEATURES,
features As Object,
ByRef size As Integer
) As Integer
```

VBImageToHandle

Converts image array to handle. This function is useful when image must be passed to PictureBox control.

Visual Basic:

```
Public Const VF_PALLETE_GREEN = 0
Public Const VF_PALLETE_GRAY = 1
Public Declare Function ImageToHandle
Lib "VFVBP42.dll" Alias "VBImageToHandle"
(
ByVal Width As Long,
ByVal Height As Long,
features As Variant,
ByRef handle As Long,
ByVal pallete As Long
) As Long
```

Visual Basic .Net:

```
Public Const VF_PALLETE_GREEN As Integer = 0
Public Const VF_PALLETE_GRAY As Integer = 1
Public Declare Function ImageToHandle
Lib "VFVBP42.dll" Alias "VBImageToHandle"
```

```
(
ByVal width As Integer,
ByVal height As Integer,
ByRef Image As Object,
ByRef handle As Integer,
ByVal pallete As Integer
) As Integer
```

VBHandleToImage

Converts handle to image array.

Visual Basic:

```
Public Declare Function HandleToImage
Lib "VFVBP42.dll" Alias "VBHandleToImage"
(
ByVal handle As Long,
ByRef width As Long,
ByRef height As Long,
Image As Variant
) As Long
```

Visual Basic .Net:

```
Public Declare Function HandleToImage
Lib "VFVBP42.dll" Alias "VBHandleToImage"
(
ByVal handle As Integer,
ByRef width As Integer,
ByRef height As Integer,
ByRef Image As Object
) As Integer
```

VBLoadImageFromFile

Loads image from specified file.

Visual Basic:

```
Public Declare Function LoadImageFromFile
Lib "VFVBP42.dll" Alias "VBLoadImageFromFile"
(
ByVal filename As String,
ByRef width As Long,
ByRef height As Long,
Image As Variant
) As Long
```

Visual Basic .Net:

```
Public Declare Function LoadImageFromFile Lib
```

```
"VFVBP42.dll" Alias "VBLoadImageFromFile"
(
ByVal filename As String,
ByRef width As Integer,
ByRef height As Integer,
ByRef Image As Object
) As Integer
```

VBDrawFeatures	Draws features on specified fingerprint image (this function is used only from Microsoft Access sample).

Visual Basic:

```
Public Const VF_DRAW_MINUTEA = 1
Public Const VF_DRAW_BLOCKED_ORIENTATIONS = 2
Public Const VF_DRAW_SINGULAR_POINTS = 4

Public Declare Function DrawFingerprintFeatures
Lib "VFVBP42.dll" Alias "VBDrawFeatures"
(
fstructure As VFFEATURES,
ByVal width As Long,
ByVal height As Long,
Image As Variant,
ByVal pallete As Long,
ByVal parameters As Long,
ByRef handle As Long
) As Long
```

Visual Basic .Net:

```
Public Const VF_DRAW_MINUTEA = 1
Public Const VF_DRAW_BLOCKED_ORIENTATIONS = 2
Public Const VF_DRAW_SINGULAR_POINTS = 4

Public Declare Function DrawFingerprintFeatures
Lib "VFVBP42.dll" Alias "VBDrawFeatures"
(
ByRef fstructure As VFFEATURES,
ByVal width As Long,
ByVal height As Long,
ByVal Image As Object,
ByVal pallete As Long,
ByVal parameters As Long,
ByRef handle As Long
) As Long
```

All parser DLL functions declarations are stated in `VFinger.bas` module (Visual Basic 6.0 sample).

5.1.6. Sun Java 2

Sample uses VeriFinger and ScanMan library through Java Native Interface - special wrappers (DLLs) were written for VeriFinger and ScanMan libraries. Class VeriFingerWrapper declares VeriFinger interface, which is accessible from Java programs, and class ScanMan declares ScanMan interface.

Sample demonstrates:

- Fingerprint enrollment
- Fingerprint enrollment with generalization
- Fingerprint identification (fast and full)
- Work with ScanMan library

VeriFinger Java demo was developed and tested only using Sun Java 2 SDK 1.4. Sun Java 2 SDK can be downloaded from http://java.sun.com/.

Used development environment - Eclipse (http://www.eclipse.org/). The project can be easily imported to Eclipse workspace, as `".project"` file is provided. Any other Java IDE can be used with equal success.

VeriFinger Java sample is located in `samples\Windows\VFDemo.java\` SDK subdirectory. It contains:

Directory/File	Description
`Application\`	Contains demo and it's source
`ReadMe.txt`	ReadMe file for Java sample

Please use `VFDemoJava.bat` to run VeriFinger Java demo (demo requires more stack size than Java allocates by default; bat file runs Java runtime with bigger stack size). Before running `VFDemoJava.bat` please make sure that path to `java.exe` is added to environment `"PATH"` variable.

5.2. Linux gtk

5.2.1. Description

This is a basic demo program for Linux. It creates a database in the memory from fingerprint image files and identifies a finger image by matching it with the ones in the database.

5.2.2. Running the gtk sample application

Before using VeriFinger SDK, please run "pgd".

For the program to work, you will need *gtk+-2.4.4* or newer and *libusb-0.1.8* or newer libraries. Many distributions include gtk+ and libusb. If You distribution doesn't have gtk+ You can find them along with installation instructions at http://www.gtk.org. Libusb can be downloaded from http://libusb.sourceforge.net/

To run the program execute the command

```
make run
```

. If You want to access hardware *You must have root permissions*. If gtk sample is used without root permissions, give correct right to scanner which You want to access. You can run as root

```
# chown <username> /proc/bus/usb/< bus num>/<device nr>
```

and then gtk sample can access these scanner with normal user rights.

5.3. Mac OS X cocoa

This is a basic demo program for Mac OS X. It creates a simple database in the memory from face image files or images from webcam and identifies face image by matching it with the ones in the database.

5.3.1. Running the Cocoa sample program

Before using VeriLook SDK, please run "pgd" from project directory.

For the program to work, you need Xcode 1.5 or newer and QuickTime 3 or later. Webcam drivers for MacOS X can be found here (tested with Philips SPC900NC).

To run this program you need to run VLDemo.app program in sample directory. If it doesn't work you need to run from Terminal command:

```
./run_sample
```

Chapter 6. Reference (C/C++)

This chapter contains reference of all libraries included in VeriFinger SDK 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.
VFExtractor	Provides functionality for extracting Neurotechnologija Finger Records from fingerprint images using VeriFinger algorithm.
FPScannerMan	Provides functionality for working with scanners.
VFMatcher	Provides functionality for comparing Neurotechnologija Finger Records using VeriFinger algorithm.

6.1. NCore Library

Provides infrastructure for Neurotechnologija components.

Import library (Windows): `NCore.dll.lib`.

DLL (Windows): `NCore.dll`.

Shared object (Linux): `libNCore.so`.

Requirements (Windows):

- `msvcr71.dll` (Microsoft C Runtime Library 7.1).

Modules

NCore	Provides infrastructure/basic functionality (such as memory management) for Neuro-

	technologija components.
NErrors	Defines error codes used in Neurotechnologija components.
NParameters	Provides functionality for work with parameters for Neurotechnologija components.
NTypes	Defines types and macros used in Neurotechnologija components.

6.1.1. NCore Module

Provides infrastructure/basic functionality (such as memory management) for Neurotechnologija components.

Header file: `NCore.h`.

Functions

`NAlloc`	Allocates memory block.
`NAStrLen`	Gets length of a null-terminated string of ansi chars.
`NCAlloc`	Allocates memory block with all bytes set to zero.
`NCompare`	Compares bytes in two memory blocks.
`NCompareAStr`	Compares two null-terminated strings of ansi chars.
`NCompareStr`	Compares two null-terminated strings of chars.
`NCopy`	Copies data between memory blocks.
`NCopyAStr`	Copies null-terminated string of ansi chars.
`NCopyStr`	Copies null-terminated string of chars.
`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.
`NStrLen`	Gets length of a null-terminated string of chars.

Macros

NClear	Clears memory block.

See Also

NCore Library

6.1.2. 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.
-201	N_E_LM_CONNECTION	Error during connection with LAN License Manager has occurred.

-202	N_E_LM_NO_MORE_LICENSE	Library has not been registered because registered license count achieved LAN License Manager license limit.
-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

6.1.3. NParameters Module

Provides functionality for work 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

6.1.3.1. 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

6.1.4. 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 (same as NAChar).
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.

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_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_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_LONG_MAX	Maximum value for NLong.
N_LONG_MIN	Minimum value for NLong.
N_MSVC	Defined if compiling with Microsoft Visual C++.
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_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_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_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

6.1.4.1. 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

6.1.4.2. 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

6.1.4.3. 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

6.1.4.3.1. NIndexPair.Index1 Field

First index of this NIndexPair.

```
NInt Index1;
```

See Also

NIndexPair

6.1.4.3.2. NIndexPair.Index2 Field

Second index of this NIndexPair.

```
NInt Index2;
```

See Also

NIndexPair

6.1.4.4. 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

6.1.4.4.1. NRational.Denominator Field

Denominator of this NRational.

```
NInt Denominator;
```

See Also

NRational

6.1.4.4.2. NRational.Numerator Field

Numerator of this NRational.

```
NInt Numerator;
```

See Also

NRational

6.1.4.5. 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

6.1.4.5.1. NURational.Denominator Field

Denominator of this NURational.

```
NUInt Denominator;
```

See Also

NURational

6.1.4.5.2. NURational.Numerator Field

Numerator of this NURational.

```
NUInt Numerator;
```

See Also

NURational

6.1.5. 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

6.1.5.1. 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

6.1.5.1.1. NPoint.X Field

Point coordinate on x axis.

```
NInt X;
```

See Also

NPoint

6.1.5.1.2. NPoint.Y Field

Point coordinate on y axis.

```
NInt Y;
```

See Also

NPoint

6.1.5.2. NSize structure

Structure defining rectangle size.

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

Fields

Width	Width.
Height	Height.

See Also

NGeometry

6.1.5.2.1. NSize.Width Field

Width.

```
NInt Width;
```

See Also

NSize

6.1.5.2.2. NSize.Height Field

Height.

```
NInt Height;
```

See Also

NSize

6.1.5.3. NRect structure

Structure defining a rectangle figure in 2D space.

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

Fields

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

See Also

NGeometry

6.1.5.3.1. NRect.X Field

Upper left rectangle coorner coordinate on x axis.

```
NInt X;
```

See Also

NRect

6.1.5.3.2. NRect.Y Field

Upper left rectangle coorner coordinate on y axis.

```
NInt Y;
```

See Also

NRect

6.1.5.3.3. NRect.Width Field

Rectangle width.

```
NInt Width;
```

See Also

NRect

6.1.5.3.4. NRect.Height Field

Rectangle height.

```
NInt Height;
```

See Also

NRect

6.2. NFRecord Library

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

Import library (Windows): `NFRecord.dll.lib`.

DLL (Windows): `NFRecord.dll`.

Shared object (Linux): `libNFRecord.so`.

Requirements (Windows):

- `NCore.dll`.

Requirements (Linux):

- `libNCore.so`.

Modules

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

6.2.1. 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 specified 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.
`NFRecordGetMinutiaNeighbour`	Retrieves the minutia neighbour at the specified index of the minutia at the specified index of the NFRecord.
`NFRecordGetMinutiaNeighbourCount`	Retrieves the number of minutia neighbours in the minutia at the specified index of the NFRecord.
`NFRecordGetMinutiaNeighbours`	Copies all minutia neighbours 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.
`NFRecordSetMinutiaNeighbour`	Sets a NFMinutiaNeighbour 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.

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.
NFMinutiaNeighbour	Represents a minutia neighbour in a NFRecord.
NFRecordInfo	For internal use.

Enumerations

NFImpressionType	Specifies the impression type.
NFMinutiaFormat	Specifies the minutia format.
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

6.2.1.1. 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

6.2.1.1.1. 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

6.2.1.1.2. 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

6.2.1.1.3. 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

6.2.1.2. 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

6.2.1.2.1. 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

6.2.1.2.2. 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

6.2.1.2.3. 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

6.2.1.2.4. 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

6.2.1.2.5. 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

6.2.1.3. 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

6.2.1.3.1. 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

6.2.1.3.2. 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

6.2.1.4. 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

6.2.1.5. 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

6.2.1.5.1. 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

6.2.1.5.2. 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

6.2.1.5.3. 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

6.2.1.5.4. 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

6.2.1.5.5. NFMinutia.Type Field

Type of this NFMinutia.

```
NFMinutiaType Type;
```

See Also

NFMinutia

6.2.1.5.6. 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

6.2.1.5.7. 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

6.2.1.6. 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

6.2.1.7. NFMinutiaNeighbour Structure

Represents a minutia neighbour in a NFRecord.

```
typedef struct NFMinutiaNeighbour_ { } NFMinutiaNeighbour;
```

Fields

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

See Also

NFRecord Module

6.2.1.7.1. NFMinutiaNeighbour.Index Field

Index of neighbour minutia.

```
NInt Index;
```

See Also

NFMinutiaNeighbour

6.2.1.7.2. NFMinutiaNeighbour.RidgeCount Field

Ridge count to neighbour minutia.

```
NByte RidgeCount;
```

See Also

NFMinutiaNeighbour

6.2.1.8. 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

6.2.1.9. 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

6.2.1.10. 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

6.2.1.11. 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

6.2.1.12. 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

6.2.1.13. 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

6.2.1.14. 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

6.2.1.15. 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

6.2.1.16. 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

6.2.1.17. 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

6.2.1.18. 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

6.2.1.19. 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

6.2.1.20. 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`

6.2.1.21. 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.
`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`

6.2.1.22. 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` | `NFRecordSaveToMemory`

6.2.1.23. 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

6.2.1.24. 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

6.2.1.25. 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

6.2.1.26. 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`

6.2.1.27. 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`

6.2.1.28. 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` | `NFRecordSetCoreCapacity`

6.2.1.29. 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`

6.2.1.30. 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.

Error Code	Condition
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`

6.2.1.31. 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`

6.2.1.32. 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` | `NFRecordSet-`

`DeltaCapacity`

6.2.1.33. 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`

6.2.1.34. 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`

6.2.1.35. 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`

6.2.1.36. 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` | `NFRecordSetDoubleCoreCapacity`

6.2.1.37. 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`

6.2.1.38. 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.

See Also

NFRecord Module | HNFRecord | `NFRecordSetG`

6.2.1.39. 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

6.2.1.40. 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

6.2.1.41. 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

6.2.1.42. 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

6.2.1.43. 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

6.2.1.44. 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`

6.2.1.45. 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

6.2.1.46. 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.

Error Code	Condition
	- 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.

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`

6.2.1.47. 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 -

Error Code	Condition
	`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`

6.2.1.48. 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`

6.2.1.49. 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`

6.2.1.50. 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`

6.2.1.51. 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`

6.2.1.52. NFRecordGetMinutiaNeighbour Function

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

```
NResult N_API NFRecordGetMinutiaNeighbour(
        HNFRecord hRecord,
        NInt minutiaIndex,
        NInt index,
        NFMinutiaNeighbour * pValue
);
```

Parameters

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

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 neighbour count obtained using `NFRecordGetMinutiaNeighbourCount` function.

See Also

NFRecord Module | HNFRecord | NFMinutiaNeighbour | `NFRecordGetMinutiaCount` | `NFRecordGetMinutiaNeighbourCount` | `NFRecordSetMinutiaNeighbour`

6.2.1.53. NFRecordGetMinutiaNeighbourCount Function

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

```
NResult N_API NFRecordGetMinutiaNeighbourCount(
        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 neighbours.

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

6.2.1.54. NFRecordGetMinutiaNeighbours Function

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

```
NResult N_API NFRecordGetMinutiaNeighbours(
        HNFRecord hRecord,
        NInt minutiaIndex,
        NFMinutiaNeighbour * arMinutiaNeighbours
);
```

Parameters

`hRecord`	[in] Handle to the NFRecord object.
`minutiaIndex`	[in] The index of minutia.
`arMinutiaNeighbours`	[out] Pointer to array of NFMinutiaNeighbour that receives minutia neighbours.

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 *`arMinutiaNeighbours`* is NULL.

Remarks

Array *`arMinutiaNeighbours`* points to must be large enough to receive all minutia neighbours. See `NFRecordGetMinutiaNeighbourCount` function.

See Also

NFRecord Module | HNFRecord | NFMinutiaNeighbour | `NFRecordGetMinutiaCount` | `NFRecordGetMinutiaNeighbourCount`

6.2.1.55. 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`

6.2.1.56. 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.

See Also

NFRecord Module | HNFRecord | NFPatternClass | `NFRecordSetPatternClass`

6.2.1.57. 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

6.2.1.58. 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`

6.2.1.59. 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

6.2.1.60. 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`

6.2.1.61. 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 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	*`buffer`* or *`pValue`* is NULL.
N_E_END_OF_STREAM	*`bufferSize`* is less than expected.

Error Code	**Condition**
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

6.2.1.62. 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`

6.2.1.63. 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`

6.2.1.64. 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`

6.2.1.65. 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

6.2.1.66. 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

6.2.1.67. 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

6.2.1.68. 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

6.2.1.69. 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`

6.2.1.70. 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`

6.2.1.71. 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`

6.2.1.72. 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.
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`

6.2.1.73. 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`

6.2.1.74. 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`

6.2.1.75. 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`

6.2.1.76. 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`

6.2.1.77. 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`

6.2.1.78. 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`

6.2.1.79. 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

6.2.1.80. 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`

6.2.1.81. 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`

6.2.1.82. 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`

6.2.1.83. 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`

6.2.1.84. 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`

6.2.1.85. 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.

Error Code	Condition
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`

6.2.1.86. 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.

See Also

NFRecord Module | HNFRecord | `NFRecordGetG`

6.2.1.87. 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`

6.2.1.88. 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`

6.2.1.89. 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`

6.2.1.90. 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`

6.2.1.91. NFRecordSetMinutiaNeighbour Function

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

```
NResult N_API NFRecordSetMinutiaNeighbour(
        HNFRecord hRecord,
        NInt minutiaIndex,
        NInt index,
        const NFMinutiaNeighbour * pValue
);
```

Parameters

`hRecord`	[in] Handle to the NFRecord object.
`minutiaIndex`	[in] The index of minutia.
`index`	[in] Index of minutia neighbour to set.
`pValue`	[in] Pointer to the NFMinutiaNeighbour 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.

Error Code	Condition
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 neighbour count obtained using `NFRecordGetMinutiaNeighbourCount` function.

See Also

NFRecord Module | HNFRecord | NFMinutiaNeighbour | `NFRecordGetMinutiaCount` | `NFRecordGetMinutiaNeighbourCount` | `NFRecordGetMinutiaNeighbour`

6.2.1.92. 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`

6.2.1.93. 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`

6.2.1.94. 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`

6.2.1.95. 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 | `NFRecordGetRidgeCountsType`

6.2.1.96. NFRidgeCountsType Enumeration

Specifies the type of ridge counts contained in a NFRecord.

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

Members

`nfrctEightNeighbours`	The NFRecord contains ridge counts to closest minutia in each of the eight sectors of each minutia. First sector starts at minutia angle.
`nfrctEightNeighboursWithIndexes`	The NFRecord contains ridge counts to eight neighbours of each minutia.
`nfrctFourNeighbours`	The NFRecord contains ridge counts to closest minutia in each of the four sectors of each minutia. First sector starts at minutia angle.
`nfrctFourNeighboursWithIndexes`	The NFRecord contains ridge counts to four neighbours of each minutia.
`nfrctNone`	The NFRecord does not contain ridge counts.
`nfrctUnspecified`	For internal use.

See Also

NFRecord Module

6.3. NImages Library

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

Import library (Windows): `NImages.dll.lib`.

DLL (Windows): `NImages.dll`.

Shared object (Linux): `libNImages.so`.

Requirements (Windows):

- `NCore.dll`.

Requirements (Linux):

- `libNCore.so`.

Modules

Bmp	Provides functionality for loading and saving images in BMP 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 work with image pixel format.
NRgbImage	Provides functionality for managing 24-bit RGB images.
Tiff	Provides functionality for loading images in TIFF format.

6.3.1. 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

6.3.1.1. 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`

6.3.1.2. 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`

6.3.1.3. 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`

6.3.1.4. 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`

6.3.1.5. 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` `BmpLoadImageFromHBitmap`

6.3.1.6. 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`

6.3.2. NGrayscaleImage Module

Provides functionality for managing 8-bit grayscale images.

Header file: `NGrayscaleImage.h`.

Functions

`NGrayscaleImageGetPixel`	Retrieves value of pixel at the specified coordinates 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

6.3.2.1. 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`

6.3.2.2. 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`

6.3.3. 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

6.3.3.1. 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`

6.3.3.2. 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`

6.3.3.3. 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`

6.3.3.4. 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

6.3.3.5. 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

6.3.3.6. 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 `NImageFormatGetFormatCount`.

See Also

NImageFormat Module | HNImageFormat | `NImageFormatGetFormatCount`

6.3.3.7. 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`

6.3.3.8. 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

6.3.3.9. 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`

6.3.3.10. 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 *`hImage-`*

Error Code	**Condition**
	`Format` does not support reading.

See Also

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

6.3.3.11. 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

Error Code	**Condition**
	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`

6.3.3.12. 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`

6.3.3.13. 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 *hImageFormat* 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`

6.3.3.14. 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`

6.3.4. 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

6.3.4.1. 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`

6.3.4.2. 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. - 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`

6.3.4.3. 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.

Error Code	Condition
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`

6.3.4.4. 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.
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`

6.3.4.5. 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`

6.3.4.6. 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.

Error Code	**Condition**

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`

6.3.4.7. 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`

6.3.4.8. 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`

6.3.4.9. 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`

6.3.4.10. 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`

6.3.4.11. 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

6.3.4.12. 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`

6.3.4.13. 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`

6.3.4.14. 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`

6.3.4.15. 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`

6.3.4.16. 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`

6.3.4.17. 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`*. - 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`

6.3.5. 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

6.3.5.1. 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`

6.3.6. 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

6.3.6.1. 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 `npfMonochrome`.

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`

6.3.6.2. 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 `npfMonochrome`.

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`

6.3.7. NPixelFormat Module

Provides functionality for work with image pixel format.

Header file: `NPixelFormat.h`.

Functions

`NPixelFormatGetBitsPerPixelFunc`	Used internally in NPixelFormatGetBitsPerPixel 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

6.3.7.1. 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

6.3.7.2. 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

6.3.7.2.1. NRgb.Blue Field

Blue component value of this NRgb.

```
NByte Blue;
```

See Also

NRgb Structure

6.3.7.2.2. NRgb.Green Field

Green component value of this NRgb.

```
NByte Green;
```

See Also

NRgb Structure

6.3.7.2.3. NRgb.Red Field

Red component value of this NRgb.

```
NByte Red;
```

See Also

NRgb Structure

6.3.8. NRgbImage Module

Provides functionality for managing 24-bit RGB images.

Header file: `NRgbImage.h`.

Functions

`NRgbImageGetPixel`	Retrieves value of pixel at the specified coordinates 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

6.3.8.1. 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`

6.3.8.2. 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`

6.3.9. 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

6.3.9.1. 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`

6.3.9.2. 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`

6.4. VFExtractor Library

Provides functionality for extracting Neurotechnologija Finger Records from fingerprint images using VeriFinger algorithm.

Import library (Windows): `VFExtractor.dll.lib`.

DLL (Windows): `VFExtractor.dll`.

Shared object (Linux): `libVFExtractor.so`.

Requirements (Windows):

- `NCore.dll`.
- `NFRecord.dll`.

Requirements (Linux):

- `libNCore.so`.
- `libNFRecord.so`.

Modules

VFExtractor	Provides functionality for extracting Neurotechnologija Finger Records (NFRecords) from fingerprint images using VeriFinger algorithm encapsulated in Neurotechnologija Fingerprint Features Extractor VF (VFExtractor) object.

6.4.1. VFExtractor Module

Provides functionality for extracting Neurotechnologija Finger Records (NFRecords) from fingerprint images using VeriFinger algorithm encapsulated in Neurotechnologija Fingerprint Features Extractor VF (VFExtractor) object.

Header file: `VFExtractor.h` (includes `VFExtractorParams.h` and `VFExtractorTypes.h`).

Functions

`VfeCopyParameters`	Copies parameter values from one VFExtractor to another.
`VfeCreate`	Creates a VFExtractor.
`VfeExtract`	Extracts a packed NFRecord from the image using the specified VFExtractor.

`VfeExtractFromImage`	Extracts a packed NFRecord from the NImage using the specified VFExtractor.
`VfeExtractUnpacked`	Extracts a NFRecord from the image using the specified VFExtractor.
`VfeExtractUnpackedFromImage`	Extracts a NFRecord from the NImage using the specified VFExtractor.
`VfeFree`	Deletes the VFExtractor. After the object is deleted the specified handle is no longer valid.
`VfeGeneralize`	Generalizes count features collections to single features collection.
`VfeGeneralizeUnpacked`	Generalizes count features collections to single features collection.
`VfeGetMaxTemplateSize`	Retrieves maximal size of packed NFRecord the specified VFExtractor can extract.
`VfeGetParameter`	Retrieves value of the specified parameter of the specified VFExtractor.
`VfeIsRegistered`	Checks if VFExtractor library is registered.
`VfeReset`	Sets default values for all parameters of the specified VFExtractor.
`VfeSetParameter`	Sets value of the specified parameter of the specified VFExtractor.

Enumerations

VfeReturnedImage	Specifies kind of image returned after extraction using VFExtractor.
VfeTemplateSize	

Types

HVFExtractor	Handle to VFExtractor object.

Macros

VFE_MODE_ATMEL_FINGERCHIP	The mode for Atmel FingerChip sensor.
VFE_MODE_AUTHENTEC_AES2501B	The mode for Authentec AES2501B sensor.
VFE_MODE_AUTHENTEC_AES4000	The mode for Authentec AES4000 sensor.
VFE_MODE_AUTHENTEC_AFS2	The mode for Authentec AF-S2 sensor.
VFE_MODE_BIOMETRIKA_FX2000	The mode for Biometrika FX2000 scanner.
VFE_MODE_BIOMETRIKA_FX3000	The mode for Biometrika FX3000 scanner.
VFE_MODE_BMF_BLP100	The mode for BMF BLP100 scanner.
VFE_MODE_CROSSMATCH_VERIFIER300	The mode for CrossMatch Verifier 300 scanner.
VFE_MODE_DIGENT_IZZIX	The mode for Digent Izzix scanner.
VFE_MODE_DIGITALPERSONA_UAREU	The mode for Digital Persona U.are.U scanner.
VFE_MODE_ETHENTICA	The mode for Ethentica scanner.
VFE_MODE_FUJITSU_MBF200	The mode for Fujitsu MBF200 scanner.
VFE_MODE_FUTRONIC_FS80	The mode for Futronic's FS80 scanner.
VFE_MODE_GENERAL	The general mode.
VFE_MODE_IDENTICATORTECHNOLOGY_DF90	The mode for Identicator Technology DF90 scanner.
VFE_MODE_IDENTIX_DFR2090	The mode for Identix DFR2090 scanner.
VFE_MODE_IDENTIX_TOUCHVIEW	The mode for Identix TouchView scanner.
VFE_MODE_KEYTRONIC_SECUREDESKTOP	The mode for Keytronic Secure Desktop scanner.
VFE_MODE_LIGHTUNING_LTTC500	The mode for LighTuning LTT-C500 scanner.
VFE_MODE_NITGEN_FINGKEY_HAMSTER	The mode for NITGEN Fingkey Hamster scanner.
VFE_MODE_PRECISEBIOMETRICS_100CS	The mode for Precise Biometrics 100CS scanner.
VFE_MODE_SECUGEN_HAMSTER	The mode for Secugen Hamster scanner.
VFE_MODE_STARTEK_FM200	The mode for Startek FM200 sensor.
VFE_MODE_TACOMA_CMOS	The mode for Tacoma CMOS sensor.

VFE_MODE_TESTECH_BIOI	The mode for Testech Bio-i sensor.
VFE_MODE_UPEK_TOUCHCHIP	The mode for UPEK TouchChip sensor.
VFEP_COPYRIGHT	Identifier specifying library copyright static read-only parameter of type N_TYPE_STRING.
VFEP_GENERALIZATION_MAXIMAL_ROTATION	Maximal rotation of two features collection to each other. Must be in range 0°..180°.
VFEP_GENERALIZATION_THRESHOLD	Has the same meaning for features generalization as VFMP_MATCHING_THRESHOLD parameter for features matching.
VFEP_MODE	Identifier specifying mode (parameter value set) parameter of type N_TYPE_UINT. Parameter value can be one of the VFE_MODE_XXX.
VFEP_NAME	Identifier specifying library name static read-only parameter of type N_TYPE_STRING.
VFEP_QUALITY_THRESHOLD	Identifier specifies image quality threshold.
VFEP_RETURNED_IMAGE	Identifier specifying kind of image returned after extraction parameter of type N_TYPE_INT. Parameter value can be one of the VfeReturnedImage enumeration members.
VFEP_TEMPLATE_SIZE	Identifier specifying template size parameter. Parameter value can be one of the VfeTemplateSize enumeration members.
VFEP_USE_QUALITY	
VFEP_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.
VFEP_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

VFExtractor Library

6.4.1.1. VfeCopyParameters Function

Copies parameter values from one VFExtractor to another.

```
NResult N_API VfeCopyParameters(
        HVFExtractor hDstExtractor,
        HVFExtractor hSrcExtractor
);
```

Parameters

hDstExtractor	[in] Handle to the destination VFExtractor object.
hSrcExtractor	[in] Handle to the source VFExtractor 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

VFExtractor Module | HVFExtractor

6.4.1.2. VfeCreate Function

Creates a VFExtractor.

```
NResult N_API VfeCreate(
        HVFExtractor * pHExtractor
);
```

Parameters

`pHExtractor`	[out] Pointer to HVFExtractor that receives handle to created VFExtractor 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 `VfeFree` function.

See Also

VFExtractor Module | HVFExtractor | `VfeFree`

6.4.1.3. VfeExtract Function

Extracts a packed NFRecord from the image using the specified VFExtractor.

```
NResult N_API VfeExtract(
        HVFExtractor 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 VFExtractor 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.

Remarks

position and *impressionType* are written to extracted NFRecord.

Memory for *buffer* must be allocated before calling the function. `VfeGetMaxTemplateSize` function should be used to learn the size needed for allocation.

Image is low quality if through *buffer* returns NULL and *pSize* - zero.

See Also

VFExtractor Module | HVFExtractor | `VfeGetMaxTemplateSize` | `VfeExtractUnpacked` | `VfeExtractFromImage`

6.4.1.4. VfeExtractFromImage Function

Extracts a packed NFRecord from the NImage using the specified VFExtractor.

```
NResult N_API VfeExtractFromImage(
        HVFExtractor hExtractor,
        HNImage hImage,
        NFPosition position,
        NFImpressionType impressionType,
        void * buffer,
        NSizeType bufferSize,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the VFExtractor 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.

Remarks

`position` and *`impressionType`* are written to extracted NFRecord.

Memory for *`buffer`* must be allocated before calling the function. `VfeGetMaxTemplateSize` function should be used to learn the size needed for allocation.

Image is low quality if through *`buffer`* returns NULL and *`pSize`* - zero.

See Also

VFExtractor Module | HVFExtractor | HNImage | `VfeGetMaxTemplateSize` | `VfeEx-`

`tract` | `VfeExtractUnpackedFromImage`

6.4.1.5. VfeExtractUnpacked Function

Extracts a NFRecord from the image using the specified VFExtractor.

```
NResult N_API VfeExtractUnpacked(
        HVFExtractor 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 VFExtractor 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.

Remarks

position and *impressionType* are written to extracted NFRecord.

Image is low quality if through *pHRecord* returns NULL.

See Also

VFExtractor Module | HVFExtractor | HNFRecord | `VfeExtract` | `VfeExtractUnpackedFromImage`

6.4.1.6. VfeExtractUnpackedFromImage Function

Extracts a NFRecord from the NImage using the specified VFExtractor.

```
NResult N_API VfeExtractUnpackedFromImage(
        HVFExtractor hExtractor,
        HNImage hImage,
        NFPosition position,
        NFImpressionType impressionType,
        HNFRecord * pHRecord
);
```

Parameters

`hExtractor`	[in] Handle to the VFExtractor 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.

Remarks

`position` and `impressionType` are written to extracted NFRecord.

Image is low quality if through `buffer` returns NULL and `pSize` - zero.

See Also

VFExtractor Module | HVFExtractor | HNImage | HNFRecord | `VfeExtractFromImage` | `VfeExtractUnpacked`

6.4.1.7. VfeFree Function

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

```
void N_API VfeFree(
        HVFExtractor hExtractor
);
```

Parameters

hExtractor	[in] Handle to the VFExtractor object.

Remarks

If *hExtractor* is NULL, does nothing.

See Also

VFExtractor Module | HVFExtractor

6.4.1.8. VfeGeneralize Function

Generalizes count features collections to single features collection.

```
NResult N_API VfeGeneralize(
        HVFExtractor hExtractor,
        NInt templateCount,
        const void * * arTemplates,
        const NSizeType * arTemplateSizes,
        void * buffer,
        NSizeType bufferSize,
        NSizeType * pSize,
        NInt * pBaseTemplateIndex
);
```

Parameters

hExtractor	[in] Handle to the VFExtractor 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 methods can not generalize templates, function returns through *pSize* zero and through *buffer* - NULL.

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.
N_E_OUT_OF_MEMORY	There was not enough memory.

See Also

VFExtractor Module | HVFExtractor | HNFRecord | `VfeExtract` | `VfeGeneralizeUnpacked`

6.4.1.9. VfeGeneralizeUnpacked Function

Generalizes count features collections to single features collection.

```
NResult N_API VfeGeneralizeUnpacked(
        HVFExtractor hExtractor,
        NInt templateCount,
        const void * * arTemplates,
        const NSizeType * arTemplateSizes,
        HNFRecord * pHRecord,
        NInt * pBaseTemplateIndex
);
```

Parameters

`hExtractor`	[in] Handle to the VFExtractor 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 methods can not generalize templates, function returns through *`pHRecord`* - NULL.

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.

See Also

VFExtractor Module | HVFExtractor | HNFRecord | `VfeExtract` | `VfeGeneralize`

6.4.1.10. VfeGetMaxTemplateSize Function

Retrieves maximal size of packed NFRecord the specified VFExtractor can extract.

```
NResult N_API VfeGetMaxTemplateSize(
        HVFExtractor hExtractor,
        NSizeType * pSize
);
```

Parameters

hExtractor	[in] Handle to the VFExtractor 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

VFExtractor Module | HVFExtractor

6.4.1.11. VfeGetParameter Function

Retrieves value of the specified parameter of the specified VFExtractor.

```
NResult N_API VfeGetParameter(
        HVFExtractor hExtractor,
        NUInt parameterId,
        void * pValue
);
```

Parameters

hExtractor	[in] Handle to the VFExtractor 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 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`:

- VFEP_COPYRIGHT
- VFEP_GENERALIZATION_THRESHOLD
- VFEP_GENERALIZATION_MAXIMAL_ROTATION
- VFEP_MODE
- VFEP_NAME
- VFEP_TEMPLATE_SIZE
- VFEP_RETURNED_IMAGE
- VFEP_VERSION_HIGH
- VFEP_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

VFExtractor Module | HVFExtractor | `VfeSetParameter`

6.4.1.12. VfeIsRegistered Function

Checks if VFExtractor library is registered.

```
NBool N_API VfeIsRegistered(void);
```

Return Values

NTrue if library is registered, NFalse otherwise.

See Also

VFExtractor Module

6.4.1.13. VfeReset Function

Sets default values for all parameters of the specified VFExtractor.

```
NResult N_API VfeReset(
        HVFExtractor hExtractor
);
```

Parameters

`hExtractor`	[in] Handle to the VFExtractor 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

VFExtractor Module | HVFExtractor

6.4.1.14. VfeReturnedImage Enumeration

Specifies kind of image returned after extraction using VFExtractor.

```
typedef enum VfeReturnedImage_ { } VfeReturnedImage;
```

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

VFExtractor Module

6.4.1.15. VfeSetParameter Function

Sets value of the specified parameter of the specified VFExtractor.

```
NResult N_API VfeSetParameter(
        HVFExtractor hExtractor,
        NUInt parameterId,
        const void * pValue
);
```

Parameters

`hExtractor`	[in] Handle to the VFExtractor 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_PARAMETER	Parameter ID is invalid.

Remarks

The following values can be used for *`parameterId`*:

- VFEP_GENERALIZATION_THRESHOLD
- VFEP_GENERALIZATION_MAXIMAL_ROTATION
- VFEP_MODE
- VFEP_RETURNED_IMAGE
- VFEP_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

VFExtractor Module | HVFExtractor | `VfeGetParameter`

6.4.1.16. VfeTemplateSize Enumeration

```
typedef enum VfeTemplateSize_ { } VfeTemplateSize;
```

Members

`vfetsLarge`	
`vfetsSmall`	

See Also

VFExtractor Module

6.5. FPScannerMan Library

Provides functionality for working with scanners.

Import library (Windows): `FPScannerMan.dll.lib`.

DLL (Windows): `FPScannerMan.dll`.

Requirements (Windows):

- `NCore.dll`.
- `NImages.dll`.

Modules

FPScanner	One instance represents one physical device.
FPScannerMan	Scanners manager enumerates, creates scanners.

6.5.1. FPScanner Module

One instance represents one physical device.

Header file: `FPScanner.h` (includes `NImage.h`).

Functions

`FPScannerCallback`	
`FPScannerGetID`	Gets associated device identifier.
`FPScannerImageScannedCallback`	
`FPScannerIsCapturing`	Checks scanner status.
`FPScannerSetFingerPlacedCallback`	
`FPScannerSetFingerRemovedCallback`	
`FPScannerSetImageScannedCallback`	
`FPScannerSetIsCapturingChangedCallback`	
`FPScannerStartCapturing`	Starts capturing fingerprint image from certain device.
`FPScannerStartCapturingForOneImage`	Starts capturing one fingerprint image from certain device.
`FPScannerStopCapturing`	Stops capturing fingerprint image.

Types

HFPScanner	Handle to HFPScanner object.

6.5.1.1. FPScannerCallback Function

```
typedef void (N_CALLBACK * FPScannerCallback)(
    HFPScanner  hScanner,
    void * pParam
);
```

Parameters

`hScanner`	[in] Handle to the FPScanner object.
`pParam`	

See Also

HFPScanner

6.5.1.2. FPScannerGetID Function

Gets associated device identifier.

```
NResult N_API FPScannerGetID(
    HFPScanner hScanner,
    NChar * pValue
);
```

Parameters

`hScanner`	[in] Handle to the FPScanner object.
`pValue`	[out] Pointer to string that receives scanner identifier.

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	`hScanner` is NULL.

See Also

HFPScanner

6.5.1.3. FPScannerImageScannedCallback Function

```
typedef void (N_CALLBACK * FPScannerImageScannedCallback)(
    HFPScanner hScanner,
    HNImage hImage,
    void * pParam
);
```

Parameters

hScanner	[in] Handle to the FPScanner object.
hImage	[in] Handle to image.
pParam	

See Also

HFPScanner | HNImage

6.5.1.4. FPScannerIsCapturing Function

Checks scanner status.

```
NResult N_API FPScannerIsCapturing(
    HFPScanner hScanner,
    NBool * pValue
);
```

Parameters

hScanner	[in] Handle to the FPScanner object.
pValue	[out] Pointer to NBool that receives value indicating whether scanner is already capturing.

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	`hScanner` or `pValue` is NULL.

See Also

HFPScanner

6.5.1.5. FPScannerSetFingerPlacedCallback Function

```
NResult N_API FPScannerSetFingerPlacedCallback(
    HFPScanner hScanner,
    FPScannerCallback pCallback,
    void * pParam
);
```

Parameters

hScanner	[in] Handle to the FPScanner object.
pCallback	
pParam	

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	`hScanner` is NULL.

See Also

HFPScanner | FPScannerCallback

6.5.1.6. FPScannerSetFingerRemovedCallback Function

```
NResult N_API FPScannerSetFingerRemovedCallback(
    HFPScanner hScanner,
    FPScannerCallback pCallback,
    void * pParam
);
```

Parameters

hScanner	[in] Handle to the FPScanner object.
pCallback	
pParam	

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	*hScanner* is NULL.

See Also

HFPScanner | FPScannerCallback

6.5.1.7. FPScannerSetImageScannedCallback Function

```
NResult N_API FPScannerSetImageScannedCallback(
    HFPScanner hScanner,
    FPScannerImageScannedCallback pCallback,
    void * pParam
);
```

Parameters

hScanner	[in] Handle to the FPScanner object.
pCallback	
pParam	

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	*`hScanner`* is NULL.

See Also

HFPScanner | FPScannerImageScannedCallback

6.5.1.8. FPScannerSetIsCapturingChangedCallback Function

```
NResult N_API FPScannerSetIsCapturingChangedCallback(
    HFPScanner hScanner,
    FPScannerCallback pCallback,
    void * pParam
);
```

Parameters

`hScanner`	[in] Handle to the FPScanner object.
`pCallback`	
`pParam`	

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	*`hScanner`* is NULL.

See Also

HFPScanner | FPScannerCallback

6.5.1.9. FPScannerStartCapturing Function

Starts capturing fingerprint image from certain device.

```
NResult N_API FPScannerStartCapturing(
    HFPScanner hScanner
);
```

Parameters

hScanner	[in] Handle to the FPScanner 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	*hScanner* or *pValue* is NULL.
N_E_FAILED	Unspecified error has occurred.
N_E_INVALID_OPERATION	The scanner is already capturing.
N_E_NOT_REGISTERED	Module is not registered.

See Also

HFPScanner

6.5.1.10. FPScannerStartCapturingForOneImage Function

Starts capturing one fingerprint image from certain device.

```
NResult N_API FPScannerStartCapturingForOneImage(
    HFPScanner hScanner
);
```

Parameters

hScanner	[in] Handle to the FPScanner 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	`hScanner` or `pValue` is NULL.
N_E_FAILED	Unspecified error has occurred.
N_E_INVALID_OPERATION	The scanner is already capturing.
N_E_NOT_REGISTERED	Module is not registered.

See Also

HFPScanner

6.5.1.11. FPScannerStopCapturing Function

Stops capturing fingerprint image.

```
NResult N_API FPScannerStopCapturing(
    HFPScanner hScanner
);
```

Parameters

`hScanner`	[in] Handle to the FPScanner 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	`hScanner` is NULL.

See Also

HFPScanner

6.5.2. FPScannerMan Module

Scanners manager enumerates, creates scanners.

Header file: `FPScannerMan.h` (includes `FPScanner.h`).

Functions

`FPScannerManGetScanner`	Retrieves the scanner at the specified index.
`FPScannerManGetScannerByID`	Retrieves the scanner by specified identifier.
`FPScannerManGetScannerCount`	Retrieves the number of scanners.
`FPScannerManInitialize`	Initializes ScannerMan library.
`FPScannerManIsRegistered`	Checks if ScannerMan library is registered.
`FPScannerManScannerCallback`	
`FPScannerManSetScannerAddedCallback`	
`FPScannerManSetScannerRemovedCallback`	
`FPScannerManUninitialize`	Uninitializes ScannerMan library.

6.5.2.1. FPScannerManGetScanner Function

Retrieves the scanner at the specified index.

```
NResult N_API FPScannerManGetScanner(
    NInt index,
    HFPScanner * pValue
);
```

Parameters

`index`	[in] Index of scanner to retrieve.
`pValue`	[out] Points to HFPScanner that receives handle to FPScanner.

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_INVALID_OPERATION	Zero initialized FPScannerMan objects.

See Also

HFPScanner

6.5.2.2. FPScannerManGetScannerByID Function

Retrieves the scanner by specified identifier.

```
NResult N_API FPScannerManGetScannerByID(
    const NChar * szID,
    HFPScanner * pValue
);
```

Parameters

`szID`	[in] Points to string that specifies the scanner.
`pValue`	[out] Points to HFPScanner that receives handle to FPScanner.

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_INVALID_OPERATION	Zero initialized FPScannerMan objects.

See Also

HFPScanner

6.5.2.3. FPScannerManGetScannerCount Function

Retrieves the number of scanners.

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

Parameters

pValue	[out] Pointer to NInt that receives number of scanners.

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_INVALID_OPERATION	Zero initialized FPScannerMan objects.

6.5.2.4. FPScannerManInitialize Function

Initializes ScannerMan library.

```
NResult N_API FPScannerManInitialize(void);
```

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_FAILED	Unspecified error has occurred.

6.5.2.5. FPScannerManIsRegistered Function

Checks if ScannerMan library is registered.

```
NBool N_API FPScannerManIsRegistered(void);
```

Return Values

NTrue if library is registered, NFalse otherwise.

6.5.2.6. FPScannerManScannerCallback Function

```
typedef void (N_CALLBACK * FPScannerManScannerCallback)(
    const NChar * szScannerID,
    void * pParam
);
```

Parameters

szScannerID	
pParam	

6.5.2.7. FPScannerManSetScannerAddedCallback Function

```
NResult N_API FPScannerManSetScannerAddedCallback(
    FPScannerManScannerCallback pCallback,
    void * pParam
);
```

Parameters

pCallback	
pParam	

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_INVALID_OPERATION	FPScannerMan is not initialized.

See Also

FPScannerManScannerCallback

6.5.2.8. FPScannerManSetScannerRemovedCallback Function

```
NResult N_API FPScannerManSetScannerRemovedCallback(
    FPScannerManScannerCallback  pCallback,
    void * pParam
);
```

Parameters

pCallback	
pParam	

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_INVALID_OPERATION	FPScannerMan is not initialized.

See Also

FPScannerManScannerCallback

6.5.2.9. FPScannerManUninitialize Function

Uninitializes ScannerMan library.

```
void N_API FPScannerManUninitialize(void);
```

See Also

FPScannerManInitialize

6.6. VFMatcher Library

Provides functionality for comparing Neurotechnologija Finger Records using VeriFinger algorithm.

Import library (Windows): `VFMatcher.dll.lib`.

DLL (Windows): `VFMatcher.dll`.

Shared object (Linux): `libVFMatcher.so`.

Requirements (Windows):

- `NCore.dll`.
- `NFRecord.dll`.

Requirements (Linux):

- `libNCore.so`.
- `libNFRecord.so`.

Modules

VFMatcher	Provides functionality for comparing Neurotechnologija Finger Records (NFRecords) using VeriFinger algorithm encapsulated in Neurotechnologija Finger Matcher VF (VFMatcher) object.

6.6.1. VFMatcher Module

Provides functionality for comparing Neurotechnologija Finger Records (NFRecords) using VeriFinger algorithm encapsulated in Neurotechnologija Finger Matcher VF (VFMatcher) object.

Header file: `VFMatcher.h` (includes `VFMatcherParams.h` and `VfmMatchDetails.h`).

Functions

`VfmCopyParameters`	Copies parameter values from one VFMatcher to another.
`VfmCreate`	Creates a VFMatcher.
`VfmFree`	Deletes the VFMatcher. After the object is deleted the specified handle is no longer valid.

`VfmGetParameter`	Retrieves value of the specified parameter of the specified VFMatcher.
`VfmIdentifyEnd`	Ends identification using the specified VFMatcher.
`VfmIdentifyNext`	Compares the specified packed NFRecord with the one identification was started with using the specified VFMatcher.
`VfmIdentifyStart`	Starts identification with the specified packed NFRecord using the specified VFMatcher.
`VfmIsRegistered`	Checks if VFMatcher library is registered.
`VfmMatchDetailsFree`	Deletes VfmMatchDetails.
`VfmReset`	Sets default values for all parameters of the specified VFMatcher.
`VfmSetParameter`	Sets value of the specified parameter of the specified VFMatcher.
`VfmVerify`	Compares two packed NFRecords using the specified VFMatcher.

Structures

VfmMatchDetails	Represents details of matching performed with VFMatcher.

Types

HVFMatcher	Handle to VFMatcher object.

See Also

VFMatcher Library

Macros

VFM_MODE_ATMEL_FINGERCHIP	The mode for Atmel FingerChip sensor.
VFM_MODE_AUTHENTEC_AES2501B	The mode for Authentec AES2501B sensor.

VFM_MODE_AUTHENTEC_AES4000	The mode for Authentec AES4000 sensor.
VFM_MODE_AUTHENTEC_AFS2	The mode for Authentec AF-S2 sensor.
VFM_MODE_BIOMETRIKA_FX2000	The mode for Biometrika FX2000 scanner.
VFM_MODE_BIOMETRIKA_FX3000	The mode for Biometrika FX3000 scanner.
VFM_MODE_BMF_BLP100	The mode for BMF BLP100 scanner.
VFM_MODE_CROSSMATCH_VERIFIER300	The mode for CrossMatch Verifier 300 scanner.
VFM_MODE_DIGENT_IZZIX	The mode for Digent Izzix scanner.
VFM_MODE_DIGITALPERSONA_UAREU	The mode for Digital Persona U.are.U scanner.
VFM_MODE_ETHENTICA	The mode for Ethentica scanner.
VFM_MODE_FUJITSU_MBF200	The mode for Fujitsu MBF200 scanner.
VFM_MODE_FUTRONIC_FS80	The mode for Futronic's FS80 scanner.
VFM_MODE_GENERAL	The general mode.
VFM_MODE_IDENTICATORTECHNOLOGY_DF90	The mode for Identicator Technology DF90 scanner.
VFM_MODE_IDENTIX_DFR2090	The mode for Identix DFR2090 scanner.
VFM_MODE_IDENTIX_TOUCHVIEW	The mode for Identix TouchView scanner.
VFM_MODE_KEYTRONIC_SECUREDESKTOP	The mode for Keytronic Secure Desktop scanner.
VFM_MODE_LIGHTUNING_LTTC500	The mode for LighTuning LTT-C500 scanner.
VFM_MODE_NITGEN_FINGKEY_HAMSTER	The mode for NITGEN Fingkey Hamster scanner.
VFM_MODE_PRECISEBIOMETRICS_100CS	The mode for Precise Biometrics 100CS scanner.
VFM_MODE_SECUGEN_HAMSTER	The mode for Secugen Hamster scanner.
VFM_MODE_STARTEK_FM200	The mode for Startek FM200 sensor.
VFM_MODE_TACOMA_CMOS	The mode for Tacoma CMOS sensor.
VFM_MODE_TESTECH_BIOI	The mode for Testech Bio-i sensor.
VFM_MODE_UPEK_TOUCHCHIP	The mode for UPEK TouchChip sensor.

VFMP_COPYRIGHT	Identifier specifying library copyright static read-only parameter of type N_TYPE_STRING.
VFMP_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%).
VFMP_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).
VFMP_MODE	Identifier specifying mode (parameter value set) parameter of type N_TYPE_UINT. Parameter value can be one of the VFM_MODE_XXX.
VFMP_NAME	Identifier specifying library name static read-only parameter of type N_TYPE_STRING.
VFMP_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.
VFMP_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.

6.6.1.1. VfmCopyParameters Function

Copies parameter values from one VFMatcher to another.

```
NResult N_API VfmCopyParameters(
        HVFMatcher hDstMatcher,
        HVFMatcher hSrcMatcher
);
```

Parameters

`hDstMatcher`	[in] Handle to the destination VFMatcher object.
`hSrcMatcher`	[in] Handle to the source VFMatcher 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

VFMatcher Module | HVFMatcher

6.6.1.2. VfmCreate Function

Creates a VFMatcher.

```
NResult N_API VfmCreate(
        HVFMatcher * pHMatcher
);
```

Parameters

`pHMatcher`	[out] Pointer to HVFMatcher that receives handle to created VFMatcher 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.
N_E_OUT_OF_MEMORY	There was not enough memory.

Remarks

Created object must be deleted using `VfmFree` function.

See Also

VFMatcher Module | HVFMatcher | `VfmFree`

6.6.1.3. VfmFree Function

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

```
void N_API VfmFree(
        HVFMatcher hMatcher
);
```

Parameters

`hMatcher`	[in] Handle to VFMatcher object.

Remarks

If `hMatcher` is NULL, does nothing.

See Also

VFMatcher Module | HVFMatcher

6.6.1.4. VfmGetParameter Function

Retrieves value of the specified parameter of the specified VFMatcher.

```
NResult N_API VfmGetParameter(
        HVFMatcher hMatcher,
        NUInt parameterId,
        void * pValue
);
```

Parameters

`hMatcher`	[in] Handle to the VFMatcher 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`*:

- VFMP_COPYRIGHT
- VFMP_MATCHING_THRESHOLD
- VFMP_MAXIMAL_ROTATION
- VFMP_MODE
- VFMP_NAME
- VFMP_VERSION_HIGH
- VFMP_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

VFMatcher Module | HVFMatcher | `VfmSetParameter`

6.6.1.5. VfmIdentifyEnd Function

Ends identification using the specified VFMatcher.

```
NResult N_API VfmIdentifyEnd(
        HVFMatcher hMatcher
);
```

Parameters

`hMatcher`	[in] Handle to the VFMatcher 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

VFMatcher Module | HVFMatcher | `VfmIdentifyStart` | `VfmIdentifyNext`

6.6.1.6. VfmIdentifyNext Function

Compares the specified packed NFRecord with the one identification was started with using the specified VFMatcher.

```
NResult N_API VfmIdentifyNext(
        HVFMatcher hMatcher,
        const void * templ,
        NSizeType templSize,
        VfmMatchDetails * pMatchDetails,
        NInt * pScore
);
```

Parameters

`hMatcher`	[in] Handle to the VFMatcher object.
`templ`	[in] Pointer to memory buffer containing packed NFRecord.
`templSize`	[in] Size of packed NFRecord.
`pMatchDetails`	[in, out] Pointer to VfmMatchDetails 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 VFMP_MATCHING_THRESHOLD and `VfmSetParameter` function), and is greater than or equal to matching threshold otherwise.

If `pMatchDetails` is not NULL it should be a pointer obtained using `VfmIdentifyStart` function.

See Also

VFMatcher Module | HVFMatcher | VfmMatchDetails | VFMP_MATCHING_THRESHOLD | `VfmSetParameter` | `VfmIdentifyStart` | `VfmIdentifyEnd`

6.6.1.7. VfmIdentifyStart Function

Starts identification with the specified packed NFRecord using the specified VFMatcher.

```
NResult N_API VfmIdentifyStart(
        HVFMatcher hMatcher,
        const void * templ,
        NSizeType templSize,
        VfmMatchDetails * * ppMatchDetails
);
```

Parameters

`hMatcher`	[in] Handle to the VFMatcher object.
`templ`	[in] Pointer to memory buffer containing packed NFRecord.
`templSize`	[in] Size of packed NFRecord.
`ppMatchDetails`	[in] Pointer to pointer to VfmMatchDetails 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 susequently passed to `VfmIdentifyNext` function to obtain details of the matching.

Finally match details should be deleted using `VfmMatchDetailsFree` function.

See Also

VFMatcher Module | HVFMatcher | VfmMatchDetails | `VfmMatchDetailsFree` | `VfmIdentifyNext` | `VfmIdentifyEnd`

6.6.1.8. VfmIsRegistered Function

Checks if VFMatcher library is registered.

```
NBool N_API VfmIsRegistered(void);
```

Return Values

NTrue if library is registered, NFalse otherwise.

See Also

VFMatcher Module

6.6.1.9. VfmMatchDetails Structure

Represents details of matching performed with VFMatcher.

```
typedef struct VfmMatchDetails_ { } VfmMatchDetails;
```

Fields

`CenterX`	X rotation point coordinate of the second (VfmVerify or VfmIdentifyNext) matched template.
`CenterY`	Y rotation point coordinate of the second (VfmVerify or VfmIdentifyNext) 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

VFMatcher Module

6.6.1.9.1. VfmMatchDetails.CenterX Field

X rotation point coordinate of the second (VfmVerify or VfmIdentifyNext) matched template.

```
NInt CenterX;
```

See Also

VfmMatchDetails

6.6.1.9.2. VfmMatchDetails.CenterY Field

Y rotation point coordinate of the second (VfmVerify or VfmIdentifyNext) matched template.

```
NInt CenterY;
```

See Also

VfmMatchDetails

6.6.1.9.3. VfmMatchDetails.MatedMinutiaCount Field

Number of mated minutiae in first and second matched NFRecords.

```
NInt MatedMinutiaCount;
```

See Also

VfmMatchDetails

6.6.1.9.4. VfmMatchDetails.MatedMinutiae Field

Pointer to array of NIndexPair containing pairs of indexes of mated minutiae in first and second matched NFRecords.

```
NIndexPair * MatedMinutiae;
```

See Also

VfmMatchDetails | NIndexPair

6.6.1.9.5. VfmMatchDetails.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

VfmMatchDetails

6.6.1.9.6. VfmMatchDetails.Score Field

Similarity score of two matched NFRecords.

```
NInt Score;
```

See Also

VfmMatchDetails

6.6.1.9.7. VfmMatchDetails.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

VfmMatchDetails | *`Rotation`*

6.6.1.9.8. VfmMatchDetails.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

VfmMatchDetails | *`Rotation`*

6.6.1.10. VfmMatchDetailsFree Function

Deletes VfmMatchDetails.

```
void N_API VfmMatchDetailsFree(
        VfmMatchDetails * pMatchDetails
);
```

Parameters

`pMatchDetails`	[in] Pointer to VfmMatchDetails to delete.

Remarks

If *`pMatchDetails`* is NULL, does nothing.

See Also

VFMatcher Module | VfmMatchDetails

6.6.1.11. VfmReset Function

Sets default values for all parameters of the specified VFMatcher.

```
NResult N_API VfmReset(
        HVFMatcher hMatcher
);
```

Parameters

`hMatcher`	[in] Handle to VFMatcher 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

VFMatcher Module | HVFMatcher

6.6.1.12. VfmSetParameter Function

Sets value of the specified parameter of the specified VFMatcher.

```
NResult N_API VfmSetParameter(
        HVFMatcher hMatcher,
        NUInt parameterId,
        const void * pValue
);
```

Parameters

`hMatcher`	[in] Handle to the VFMatcher 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 -

Error Code	Condition
	`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.
N_E_PARAMETER_READ_ONLY	`parameterId` specifies read-only parameter.

Remarks

The following values can be used for `parameterId`:

- VFMP_MATCHING_THRESHOLD
- VFMP_MAXIMAL_ROTATION
- VFMP_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

VFMatcher Module | HVFMatcher | `VfmGetParameter`

6.6.1.13. VfmVerify Function

Compares two packed NFRecords using the specified VFMatcher.

```
NResult N_API VfmVerify(
        HVFMatcher hMatcher,
        const void * template1,
        NSizeType template1Size,
        const void * template2,
        NSizeType template2Size,
        VfmMatchDetails * * ppMatchDetalis,
        NInt * pScore
);
```

Parameters

`hMatcher`	[in] Handle to the VFMatcher 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 VfmMatchDetails 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 VFMP_MATCHING_THRESHOLD and `VfmSetParameter` 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 `VfmMatchDetailsFree` function.

See Also

VFMatcher Module | HVFMatcher | VfmMatchDetails | VFMP_MATCHING_THRESHOLD | `VfmSetParameter` | `VfmMatchDetailsFree` | `VfmIdentifyStart` | `VfmIdentifyNext` | `VfmIdentifyEnd`

Chapter 7. Reference (.NET)

This chapter contains reference of all libraries included in VeriFinger SDK for .NET developers.

C# language is used where it is needed to provide code.

Libraries

Neurotec	Provides classes that provide infrastructure for Neurotechnologija components.
Neurotec.Biometrics.FPScannerMan	Provides functionality for working with scanners.
Neurotec.Biometrics.Gui.NFView	Privides functionality for showing the image. Allows to show, move minutiae over the fingerprint image.
Neurotec.Biometrics.NFRecord	Provides functionality for packing, unpacking and editing Neurotechnologija Finger Records.
Neurotec.Biometrics.VFExtractor	Provides functionality for extracting Neurotechnologija Finger Records from fingerprint images using VeriFinger algorithm.
Neurotec.Biometrics.VFMatcher	Provides functionality for comparing Neurotechnologija Finger Records using VeriFinger algorithm.
Neurotec.Images	Provides classes that enable loading, saving and converting images in various formats.

7.1. Neurotec Library

Provides classes that provide infrastructure for Neurotechnologija components.

DLL: `Neurotec.dll`.

Namespaces

Neurotec	Contains classes that provide infrastructure for Neurotechnologija components.

7.1.1. Neurotec Namespace

Contains classes that provide infrastructure for Neurotechnologija components.

Classes

LicenceManagerException	The exception that is thrown when trying to register a Neurotechnologija library with License Manager server and an error has occurred.
NCore	This class supports internal Neurotechnologija libraries infrastructure and should not be used directly in your code.
NeurotecException	The exception that is thrown when unknown error occurred in one of Neurotechnologija libraries.
NotRegisteredException	The exception that is thrown when using unregistered Neurotechnologija library.
NParameters	This class supports internal Neurotechnologija libraries infrastructure and should not be used directly in your code.
NResult	This class supports internal Neurotechnologija libraries infrastructure and should not be used directly in your code.
ParameterException	The exception that is thrown when parameter code provided to a parameter value get or set method is not valid.
ParameterReadOnlyException	The exception that is thrown when parameter, which code is provided to a parameter value set method, is read-only.

Structures

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

Enumerations

NByteOrder	Specifies byte order.

7.1.1.1. NByteOrder Enumeration

Specifies byte order.

```
public enum NByteOrder
```

Members

Member	Description
BigEndian	Big-endian byte order.
LittleEndian	Little-endian byte order.

7.1.1.2. NIndexPair Structure

Represents pair of indexes.

Constructors

NIndexPair	Initializes a new instance of the NIndexPair structure.

Properties

Index1	Gets or sets first index of this NIndexPair.
Index2	Gets or sets second index of this NIndexPair.

7.1.1.2.1. Index1 Property

Gets or sets first index of this NIndexPair.

```
public int Index1 {get; set;}
```

Property value

First index of this NIndexPair.

7.1.1.2.2. Index2 Property

Gets or sets second index of this NIndexPair.

```
public int Index2 {get; set;}
```

Property value

Second index of this NIndexPair.

7.1.1.2.3. NIndexPair Constructor

```
public NIndexPair(
        int index1,
        int index2
);
```

Parameters

index1	First index of this NIndexPair.
index2	Second index of this NIndexPair.

7.1.1.3. NRational Structure

Represents a signed rational number.

Constructors

NRational	Initializes a new instance of the NRational structure.

Fields

Empty	Represents a NRational that is a null reference.

Properties

Denominator	Sets or retrieves the NRational value Denominator.
Numerator	Sets or retrieves the NRational value Nu-

	merator.

7.1.1.3.1. NRational Constructor

Initializes a new instance of the NRational structure.

```
public NRational(
        int numerator,
        int denominator
);
```

Parameters

`numerator`	Numerator of this NRational.
`denominator`	Denominator of this NRational.

7.1.1.3.2. Empty Field

Represents a NRational that is a null reference.

```
public static readonly NRational Empty
```

7.1.1.3.3. Denominator Property

Sets or retrieves the NRational value Denominator.

```
public int Denominator {get; set;}
```

Property value

Denominator of this NRational.

7.1.1.3.4. Numerator Property

Sets or retrieves the NRational value Numerator.

```
public int Numerator {get; set;}
```

Property value

Numerator of this NRational.

7.1.1.4. NURational Structure

Represents an unsigned rational number.

Constructors

NURational	Initializes a new instance of the NURational structure.

Fields

Empty	Represents a NURational that is a null reference.

Properties

Denominator	Sets or retrieves the NURational value Denominator.
Numerator	Sets or retrieves the NURational value Numerator.

7.1.1.4.1. NURational Constructor

Initializes a new instance of the NURational structure.

```
public NURational(
        int numerator,
        int denominator
);
```

Parameters

numerator	Numerator of this NURational.
denominator	Denominator of this NURational.

7.1.1.4.2. Empty Field

Represents a NURational that is a null reference.

```
public static readonly NURational Empty
```

7.1.1.4.3. Denominator Property

Sets or retrieves the NURational value Denominator.

```
public int Denominator {get; set;}
```

Property value

Denominator of this NURational.

7.1.1.4.4. Numerator Property

Sets or retrieves the NURational value Numerator.

```
public int Numerator {get; set;}
```

Property value

Numerator of this NURational.

7.1.1.5. NeurotecException Class

The exception that is thrown when unknown error occurred in one of Neurotechnologija libraries.

Properties

Code	Gets a error code.
Message	Gets a message that describes the current exception.

7.1.1.5.1. Code Property

Gets a error code.

```
public int Code {get;}
```

Property value

An error code.

7.1.1.5.2. Message Property

Gets a message that describes the current exception.

```
public override string Message {get;}
```

Property value

An error message.

7.2. Neurotec.Biometrics.FPScannerMan Library

Provides functionality for working with scanners.

DLL: `Neurotec.Biometrics.FPScannerMan.dll`.

Namespaces

Neurotec.Biometrics	Provides functionality for working with scanners.

7.2.1. Neurotec.Biometrics Namespace

List of VeriFinger SDK supported scanners under Windows OS could be found in Table 3.1, “Supported scanners for different platforms”

Scanner drivers for Windows OS are available in `install\Fingerprint Scanners` folder.

Classes

Class	Description
FPScanner	One instance represents one physical device.
FPScannerImageScannedEventArgs	The class contains data of event ImageScanned.
FPScannerMan	Scanners manager enumerates, creates and monitors scanners.
FPScannerMan.FPScannerCollection	Represents the collection of FPScanner
FPScannerManScannerEventArgs	The class contains data of events ScannerAdded, ScannerRemoved

Delegates

Delegate	Description
FPScannerManScannerEventHandler	Represents the method that handles the ScannerAdded and ScannerRemoved events.

Delegate	**Description**
FPScannerImageScannedEventHandler	Represents the method that handles a ImageScanned event.

7.2.1.1. FPScanner Class

One instance represents one physical device.

Properties

Handle	Gets handle to the scanner.
ID	Gets associated device identifier.
IsCapturing	Checks scanner status.

Methods

StartCapturing	Starts capturing fingerprint image from certain device.
StartCapturingForOneImage	Starts capturing one fingerprint image from certain device.
StopCapturing	Stops capturing fingerprint image.

Events

FingerPlaced	Occurs when finger is placed on scanner.
FingerRemoved	Occurs when finger is removed from scanner.
ImageScanned	Occurs when image is scanned from scanner.
IsCapturingChanged	Occurs when capturing from scanner was started, stoped for or capturing was started for one image.

7.2.1.1.1. Handle Property

Gets handle to the scanner.

```
public IntPtr Handle {get;}
```

Property value

The handle to the scanner.

7.2.1.1.2. ID Property

Gets associated device identifier.

```
public string ID {get;}
```

Property value

An associated device identifier.

7.2.1.1.3. IsCapturing Property

Checks scanner status.

```
public bool IsCapturing {get;}
```

Property value

`true` if capturing fror scanner was started; otherwise, `false`.

7.2.1.1.4. StartCapturing Method

Starts capturing fingerprint image from certain device.

```
public void StartCapturing();
```

See Also

StartCapturingForOneImage | StopCapturing

7.2.1.1.5. StartCapturingForOneImage Method

Starts capturing one fingerprint image from certain device.

```
public void StartCapturingForOneImage();
```

Remarks

Scanner is stoped automatically when fingerprint is scanned.

See Also

StartCapturingForOneImage

7.2.1.1.6. StopCapturing Method

Stops capturing fingerprint image.

```
public void StopCapturing();
```

See Also

StartCapturing | StartCapturingForOneImage

7.2.1.1.7. FingerPlaced Event

Occurs when finger is placed on scanner.

```
public event EventHandler FingerPlaced
```

See Also

FingerRemoved | ImageScanned

7.2.1.1.8. FingerRemoved Event

Occurs when finger is removed from scanner.

```
public event EventHandler FingerRemoved
```

See Also

FingerPlaced | ImageScanned

7.2.1.1.9. ImageScanned Event

Occurs when image is scanned from scanner.

```
public event FPScannerImageScannedEventHandler ImageScanned
```

See Also

FingerPlaced | FingerRemoved | FPScannerImageScannedEventHandler

7.2.1.1.10. IsCapturingChanged Event

Occurs when capturing from scanner was started, stoped for or capturing was started for one image.

```
public event EventHandler IsCapturingChanged
```

7.2.1.2. FPScannerImageScannedEventArgs Class

The class contains data of event ImageScanned.

Constructors

FPScannerImageScannedEventArgs Constructor	Initializes a new instance of the `FPScannerImageScannedEventArgs` class.

Properties

Image	Gets scanned image.

7.2.1.2.1. FPScannerImageScannedEventArgs Constructors

Initializes a new instance of the FPScannerImageScannedEventArgs class.

```
public FPScannerImageScannedEventArgs(
        NGrayscaleImage image
);
```

Parameters

`image`	The NGrayscaleImage image.

See Also

FPScanner | FPScannerImageScannedEventHandler

7.2.1.2.2. Image Property

Gets scanned image.

```
public NGrayscaleImage Image {get;}
```

Property value

The NGrayscaleImage object.

See Also

NGrayscaleImage

7.2.1.3. FPScannerMan Class

Constructors

FPScannerMan Constructor	Initializes a new instance of the `FPScannerMan` class.

Properties

IsRegistered	Checks if ScannerMan library is registered.
Scanners	Gets FPScannerMan.FPScannerCollection collection.

Methods

Dispose	Releases the resources used by `FPScannerMan`.

Events

ScannerAdded	Occurs when the scanner is connected.
ScannerRemoved	Occurs when the scanner ir unpluged.

Constants

DllName	Name of DLL containing unmanaged part of this class.

7.2.1.3.1. FPScannerMan Constructor

Initializes a new instance of the `FPScannerMan` class.

```
public FPScannerMan(
        ISynchronizeInvoke synInvoke
);
```

Parameters

`synInvoke`	An ISynchronizeInvoke object.

7.2.1.3.2. IsRegistered Property

Checks if ScannerMan library is registered.

```
public static bool IsRegistered {get;}
```

Property value

`true` if library is registered, `false` if library is not registered.

7.2.1.3.3. Scanners Property

Gets FPScannerMan.FPScannerCollection collection.

```
public FPScannerMan.FPScannerCollection Scanners {get;}
```

Property value

A FPScannerMan.FPScannerCollection collection.

See Also

FPScannerMan.FPScannerCollection

7.2.1.3.4. Dispose Method

Releases the resources used by `FPScannerMan`.

```
public void Dispose();
```

7.2.1.3.5. ScannerAdded Event

Occurs when the scanner is connected.

```
public event FPScannerManScannerEventHandler ScannerAdded
```

See Also

FPScannerManScannerEventHandler

7.2.1.3.6. ScannerRemoved Event

Occurs when the scanner ir unpluged.

```
public event FPScannerManScannerEventHandler ScannerRemoved
```

See Also

FPScannerManScannerEventHandler

7.2.1.4. FPScannerMan.FPScannerCollection Class

Represents the collection of FPScanner

Properties

Item	Gets FPScanner from collection by index.

7.2.1.4.1. FPScannerMan.FPScannerCollection.Item Property

7.2.1.4.1.1. this[int]

Gets FPScanner from collection by index.

```
public FPScanner this[
    int index
] {get;}
```

Property value

A FPScanner object.

See Also

FPScanner

7.2.1.4.1.2. this[string]

Gets FPScanner from collection by scanner ID.

```
public FPScanner this[
    string id
] {get;}
```

Property value

A FPScanner object.

See Also

FPScanner

7.2.1.4.1.3. this[IntPtr]

Gets FPScanner from collection by scanner ID.

```
public FPScanner this[
    IntPtr handle
] {get;}
```

Property value

A FPScanner object.

See Also

FPScanner

7.2.1.5. FPScannerManScannerEventArgs Class

The class contains data of events ScannerAdded, ScannerRemoved

Constructors

FPScannerManScannerEventArgs Constructor	Initializes a new instance of the `FPScannerManScannerEventArgs` class.

Properties

Scanner	Gets scanner id.

7.2.1.5.1. FPScannerManScannerEventArgs Constructor

Initializes a new instance of the `FPScannerManScannerEventArgs` class.

```
public FPScannerManScannerEventArgs(
        FPScanner scanner
);
```

Parameters

`scanner`	The FPScanner object.

See Also

FPScannerMan

7.2.1.5.2. Scanner Property

Gets the FPScanner object.

```
public FPScanner Scanner {get;}
```

Property value

The FPScanner object.

7.2.1.6. FPScannerImageScannedEventHandler Delegate

```
public delegate void FPScannerImageScannedEventHandler(
    object sender,
    FPScannerImageScannedEventArgs ea
);
```

Parameters

sender	The source of the event.
ea	A FPScannerImageScannedEventArgs that contains the event data.

See Also

7.2.1.7. FPScannerManScannerEventHandler Delegate

```
public delegate void FPScannerManScannerEventHandler(
        object sender,
        FPScannerManScannerEventArgs  e
);
```

Parameters

sender	The source of the event.
e	A FPScannerManScannerEventArgs that contains the event data.

See Also

7.3. Neurotec.Biometrics.Gui.NFView Library

Privides functionality for showing the image. Allows to show, move minutiae over the fingerprint image.

DLL: `Neurotec.Biometrics.Gui.NFView.dll`.

Namespaces

Neurotec.Biometrics.Gui	Namespace contains user interface controls.

7.3.1. Fingerprint view component

Namespace contains user interface controls.

The fingerprint view component `NFView` allows a developer to show fingerprint image. The purpose of the interface is lets a programmer manipulate `NFView` control.

Classes

Class	Description
NFView	Initializes the component.

Enumerations

Enumeration	Description
ShownImage	Specifies type of showed image.

7.3.1.1. NFView Class

Constructors

NFView Constructor	Initializes the component.

Properties

AllowHover	Allows or denies mouse hover of minutiae.
AllowSelection	Allows or denies selection of minutiae.

HoveredMinutiaIndex	Gets index of hovered minutia.
Image	Gets or sets fingerprint image.
MinutiaColor	Gets or sets minutiae color. By default is red color.
NeighbourMinutiaColor	Gets or sets neighbour minutiae color. By default orange color.
ResultImage	Bitmap binarized or skeletonized image - the binarized or skeletonized image. Gets or sets fingerprint binarized or skeletonized image.
SelectedMinutiaColor	Gets or sets selected minutiae color. By default magenta color.
SelectedMinutiaIndex	Gets or sets index of selected minutia.
ShownImage	Gets or sets one of None, Original, Result. Original - shows the image, Result - shows the binarized or skeletonized image, None - hides the image.
ShowMinutiae	Tells if minutiae should be displayed. `true` - displays minutiae, `false` - hides minutiae.
Template	Gets or sets the fingerprint template.
Zoom	Gets or sets number greater than zero for zooming the fingerprint view.

Methods

GetMinutiaAtScreenPoint	Gets index of minutia, that displayed at specified screen point.
ScreenPointToMinutiaPosition	Calculates position of minutia from given screen point.

Events

HoveredMinutiaIndexChanged	Occurs when hovered minutia index changed.
ImageChanged	Occurs when image changed.

MinutiaColorChanged	Occurs when minutia color changed.
NeighbourMinutiaColorChanged	Occurs when neighbour minutia color changed.
ResultImageChanged	Occurs when result image changed.
SelectedMinutiaColorChanged	Occurs when selected minutia color changed.
SelectedMinutiaIndexChanged	Occurs when selected minutia index changed.
ShownImageChanged	Occurs when shown image changed.
ShownMinutiaeChanged	Occurs when shown minutiae changed.
TemplateChanged	Occurs when template changed.
ZoomChanged	Occurs when zoom changed.

7.3.1.1.1. NFView Constructor

Initializes the component.

```
public NFView();
```

See Also

NFView Class

7.3.1.1.2. AllowHover Property

Allows or denies mouse hover of minutiae.

```
public bool AllowHover {get; set;}
```

Property value

`true` - enable hover, `false` - disable hover.

7.3.1.1.3. AllowSelection Property

Allows or denies selection of minutiae.

```
public bool AllowSelection {get; set;}
```

Property value

`true` - enable selection, `false` - disable selection.

7.3.1.1.4. HoveredMinutiaIndex Property

Gets index of hovered minutia.

```
public int HoveredMinutiaIndex {get;}
```

Property value

Index of hovered minutia.

See Also

SelectedMinutiaIndex

7.3.1.1.5. MinutiaColor Property

Gets or sets neighbour minutiae color. By default orange color.

```
public Color NeighbourMinutiaColor {get; set;}
```

Property value

`System.Drawing.Color` value.

See Also

MinutiaColor, SelectedMinutiaColor

7.3.1.1.6. Image Property

Gets or sets fingerprint image.

```
public Bitmap Image {get; set;}
```

Property value

A `System.Drawing.Bitmap` object.

See Also

ResultImage

7.3.1.1.7. MinutiaColor Property

Gets or sets minutiae color. By default is red color.

```
public Color MinutiaColor {get; set;}
```

Property value

`System.Drawing.Color` value.

See Also

SelectedMinutiaColor, NeighbourMinutiaColor

7.3.1.1.8. ResultImage Property

Bitmap binarized or skeletonized image - the binarized or skeletonized image. Gets or sets fingerprint binarized or skeletonized image.

```
public Bitmap ResultImage {get; set;}
```

Property value

A `System.Drawing.Bitmap` object.

See Also

Image

7.3.1.1.9. MinutiaColor Property

Gets or sets selected minutiae color. By default magenta color.

```
public Color SelectedMinutiaColor {get; set;}
```

Property value

`System.Drawing.Color` value.

See Also

MinutiaColor, NeighbourMinutiaColor

7.3.1.1.10. SelectedMinutiaIndex Property

Gets or sets index of selected minutia.

```
public int SelectedMinutiaIndex {get; set;}
```

Property value

Index of selected minutia.

See Also

HoveredMinutiaIndex

7.3.1.1.11. ShowMinutiae Property

Tells if minutiae should be displayed. `true` - displays minutiae, `false` - hides minutiae.

```
public bool ShowMinutiae {get; set;}
```

Property value

`true` - displays minutiae, `false` - hides minutiae.

See Also

ShownImage

7.3.1.1.12. ShownImage Property

Gets or sets one of None, Original, Result. Original - shows the image, Result - shows the binarized or skeletonized image, None - hides the image.

```
public ShownImage ShownImage {get; set;}
```

Property value

Original - shows the image, Result - shows the binarized or skeletonized image, None - hides the image.

See Also

ShowMinutiae

7.3.1.1.13. Template Property

Gets or sets the fingerprint template.

```
public NFRecord Template {get; set;}
```

Property value

NFRecord object.

7.3.1.1.14. Zoom Property

Gets or sets number greater than zero for zooming the fingerprint view.

```
public float Zoom {get; set;}
```

Property value

Float value. 1.0 - original image size.

7.3.1.1.15. GetMinutiaAtScreenPoint Method

Gets index of minutia, that displayed at specified screen point.

```
public int GetMinutiaAtScreenPoint(int x, int y);
```

Parameters

x, y - position in screen.

Return Values

Index of minutia. -1 if no minutia occurs at given point.

See Also

ScreenPointToMinutiaPosition

7.3.1.1.16. ScreenPointToMintiaPosition Method

Calculates position of minutia from given screen point.

```
public Point ScreenPointToMinutiaPosition(int x, int y);
```

Parameters

x, y - position in screen.

Return Values

Point structure, that contains minutia position.

See Also

GetMinutiaAtScreenPoint

7.3.1.1.17. HoveredMinutiaIndexChanged Event

Occurs when hovered minutia index changed.

```
public event EventHandler HoveredMinutiaIndexChanged;
```

7.3.1.1.18. ImageChanged Event

Occurs when image changed.

```
public event EventHandler ImageChanged;
```

7.3.1.1.19. MinutiaColorChanged Event

Occurs when minutia color changed.

```
public event EventHandler MinutiaColorChanged;
```

7.3.1.1.20. NeighbourMinutiaColorChanged Event

Occurs when neighbour minutia color changed.

```
public event EventHandler NeighbourMinutiaColorChanged;
```

7.3.1.1.21. ResultImageChanged Event

Occurs when result image changed.

```
public event EventHandler ResultImageChanged;
```

7.3.1.1.22. ShownImageChanged Event

Occurs when shown image changed.

```
public event EventHandler ShownImageChanged;
```

7.3.1.1.23. ShownMinutiaeChanged Event

Occurs when shown minutiae changed.

```
public event EventHandler ShownMinutiaeChanged;
```

7.3.1.1.24. SelectedMinutiaColorChanged Event

Occurs when selected minutia color changed.

```
public event EventHandler SelectedMinutiaColorChanged;
```

7.3.1.1.25. SelectedMinutiaIndexChanged Event

Occurs when selected minutia index changed.

```
public event EventHandler SelectedMinutiaIndexChanged;
```

7.3.1.1.26. TemplateChanged Event

Occurs when template changed.

```
public event EventHandler TemplateChanged;
```

7.3.1.1.27. ZoomChanged Event

Occurs when zoom changed.

```
public event EventHandler ZoomChanged;
```

7.3.1.2. ShownImage Enumeration

Specifies type of showed image.

Members

Member	Description
None	An image is hidden.
Original	Original image.
Result	Skeletonized or binarized image.

7.4. Neurotec.Biometrics.NFRecord Library

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

DLL: `Neurotec.Biometrics.NFRecord.dll`.

Namespaces

Neurotec.Biometrics	Provides classes for packing, unpacking and editing Neurotechnologija Finger Records.

7.4.1. Neurotec.Biometrics Namespace

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

Classes

NFRecord	Contains functionality for packing, unpacking and editing NFRecord information.
NFRecord.CoreCollection	Represents the collection of NFCore.
NFRecord.DeltaCollection	Represents the collection of NFDelta.
NFRecord.DoubleCoreCollection	Represents the collection of NFDoubleCore.
NFRecord.MinutiaCollection	Represents the collection of NFMinutia.
NFRecord.MinutiaNeighboursCollection	Represents the collection of NFMinutiaNeighbour.

Structures

NFCore	The structure contains information of core.
NFDelta	The structure contains information of delta.
NFDoubleCore	The structure contains information of double core.
NFMinutia	The structure contains information of minutia.
NFMinutiaNeighbour	The structure contains information of minutia neighbour.

Enumerations

NFImpressionType	Specifies the impression types.
NFMinutiaFormat	Specifies formats of minutia.
NFMinutiaType	Specifies types of minutia.
NFPatternClass	Specifies pattern class of the fingerprint.
NFPosition	Specifies finger position.
NFRidgeCountsType	Specifies type of ridge counts contained in NFRecord.

7.4.1.1. NFRecord Class

Constructors

NFRecord Constructor	Initializes a new instance of the `NFRecord` class.

Properties

CbeffProductType	Gets or sets the Cbeff product type of the NFRecord.
Cores	Gets cores collection.
Deltas	Gets deltas collection.
DoubleCores	Gets double cores collection.
G	Gets or sets additional fingerprint coefficient.
Handle	Gets handle to unmanaged `NFRecord` object.
Height	Gets the height of fingerprint image.
HorzResolution	Gets horizontal resolution of fingerprint image.
ImpressionType	Gets or sets the impression type of the finger record.
Minutiae	Gets minutiae collection.
MinutiaeNeighbours	Gets minutia neighbours from NFRecord.
MinutiaFormat	Gets or sets minutia format minutia format from NFRecord.
PatternClass	Gets or sets pattern class.
Position	Gets or sets finger position.
Quality	Gets or sets fingerprint quality.
RidgeCountsType	Gets or sets ridge counts type.
VertResolution	Gets vertical resolution of fingerprint image.
Width	Gets width of fingerprint image.

Methods

Check	Checks if format of packed `NFRecord` is correct.
Clone	Creates `NFRecord` object from another `NFRecord` object.
Dispose	Releases the resources used by NFRecord.
FromHandle	Creates `NFRecord` object from handle.
GetCbeffProductType	
GetG	Retrieves G from packed `NFRecord`.
GetHeight	Retrieves height of fingerprint image from packed `NFRecord`.
GetHorzResolution	Retrieves horizontal resolution of fingerprint image from packed `NFRecord`.
GetImpressionType	Retrieves impression type from packed NFRecord.
GetMaxSize	Calculates the maximal `NFRecord` size.
GetMaxSizeV1	Calculates the maximal version 1.0 `NFRecord` size.
GetPatternClass	Retrieves fingerprint pattern class from packed `NFRecord`.
GetPosition	Retrieves finger position from packed `NFRecord`.
GetQuality	Retrieves fingerprint quality from packed `NFRecord`.
GetSize	Calculates packed size of `NFRecord`.
GetSizeV1	Calculates version 1.0 packed size of NFRecord.
GetVertResolution	Retrieves vertical resolution of fingerprint image from packed `NFRecord`.
GetWidth	Retrieves width of fingerprint image from packed `NFRecord`.
Save	Packs `NFRecord` to byte array.
SaveV1	Packs `NFRecord` to memory buffer in ver-

	sion 1.0 format.

Constants

DllName	Name of DLL containing unmanaged part of this class.
FlagSaveBlockedOrients	The flag indicating whether blocked orientations should be packed in NFRecord.
FlagSkipBlockedOrients	The flag indicating whether blocked orientations should be skipped while unpacking NFRecord.
FlagSkipCurvatures	The flag indicating whether minutiae curvatures should be skipped while unpacking or packing NFRecord.
FlagSkipGs	The flag indicating whether minutiae gs should be skipped while unpacking or packing NFRecord.
FlagSkipQualities	The flag indicating whether minutiae qualities should be skipped while unpacking or packing NFRecord.
FlagSkipRidgeCounts	The flag indicating whether ridge counts should be skipped while unpacking or packing NFRecord.
FlagSkipSingularPoints	The flag indicating whether singular points (cores, deltas and double cores) should be skipped while unpacking or packing NFRecord.
MaxCoreCount	The maximum number of cores a NFRecord can contain.
MaxDeltaCount	The maximum number of deltas a NFRecord can contain.
MaxDimension	The maximum value for x and y coordinates of a minutia, core, delta or double core in a NFRecord.
MaxDoubleCoreCount	The maximum number of double cores a NFRecord can contain.
MaxMinutiaCount	The maximum number of minutiae a NFRecord can contain.

Resolution	The resolution of minutiae, cores, deltas and double cores coordinates in a NFRecord.

7.4.1.1.1. NFRecord Constructors

Initializes a new instance of the `NFRecord` class.

7.4.1.1.1.1. NFRecord(byte[], uint,out NFRecordInfo)

```
public NFRecord(
byte[] buffer,
uint flags,
out NFRecordInfo info
);
```

Parameters

`buffer`	The packed `NFRecord` object.
`flags`	Bitwise combination of zero or more flags that controls behavior of the constructor.
`info`	The `NFTemplateInfo` object.

7.4.1.1.1.2. NFRecord(byte[],out NFRecordInfo)

```
public NFRecord(
byte[] buffer,
out NFRecordInfo info
);
```

Parameters

`buffer`	The packed `NFRecord` object.
`info`	The `NFTemplateInfo` object.

7.4.1.1.1.3. NFRecord(byte[] buffer)

Initializes a new instance of the `NFRecord` class.

```
public NFRecord(
byte[] buffer
);
```

Parameters

`buffer`	The packed `NFRecord` object.

7.4.1.1.1.4. NFRecord(byte[] buffer, uint flags)

Initializes a new instance of the `NFRecord` class.

```
public NFRecord(
byte[] buffer,
uint flags
);
```

Parameters

`buffer`	The packed `NFRecord` object.
`flags`	Bitwise combination of zero or more flags that controls behavior of the constructor.

7.4.1.1.1.5. NFRecord(ushort,ushort,ushort,ushort)

```
public NFRecord(
ushort width,
ushort height,
ushort horzResolution,
ushort vertResolution
);
```

Parameters

`width`	The fingerprint image width.
`height`	The fingerprint image height.
`horzResolution`	Horizontal resolution in pixels per inch of fingerprint image.
`vertResolution`	Vertical resolution in pixels per inch of fingerprint image.

7.4.1.1.1.6. NFRecord(ushort,ushort,ushort,ushort,uint)

```
public NFRecord(
ushort width,
ushort height,
ushort horzResolution,
ushort vertResolution,
uint flags
);
```

Parameters

width	The fingerprint image width.
height	The fingerprint image height.
horzResolution	Horizontal resolution in pixels per inch of fingerprint image.
vertResolution	Vertical resolution in pixels per inch of fingerprint image.
flags	Bitwise combination of zero or more flags that controls behavior of the constructor.

7.4.1.1.2. CbeffProductType Property

Gets or sets the Cbeff product type of the NFRecord.

```
public ushort CbeffProductType {set; get;}
```

Property value

The Cbeff product type.

See Also

GetCbeffProductType

7.4.1.1.3. Cores Property

Gets cores collection.

```
public NFRecord.CoreCollection Cores {get;}
```

Property value

A NFRecord.CoreCollection that contains cores.

See Also

NFRecord.CoreCollection

7.4.1.1.4. Deltas Property

Gets deltas collection.

```
public NFRecord.DeltaCollection Deltas {get;}
```

Property value

A NFRecord.DeltaCollection that contains deltas.

See Also

NFRecord.DeltaCollection

7.4.1.1.5. DoubleCores Property

Gets doublecores collection.

```
public NFRecord.DoubleCoreCollection DoubleCores {get;}
```

Property value

A NFRecord.DoubleCoreCollection that contains double cores.

See Also

NFRecord.DoubleCoreCollection

7.4.1.1.6. G Property

Gets or sets additional fingerprint coefficient.

```
public byte G {get; set;}
```

Property value

Fingerprint coefficient.

Remarks

G - average fingerprint ridge density.

7.4.1.1.7. Handle Property

Gets handle to unmanaged `NFRecord` object.

```
public IntPtr Handle {get;}
```

Property value

A handle to unmanaged NFRecord object.

7.4.1.1.8. Height Property

Gets the height of fingerprint image.

```
public ushort Height {get;}
```

Property value

Height of fingerprint image.

See Also

Width

7.4.1.1.9. HorzResolution Property

Gets horizontal resolution of fingerprint image.

```
public ushort HorzResolution {get;}
```

Property value

Horizontal resolution in pixels per inch of fingerprint image.

See Also

VertResolution

7.4.1.1.10. ImpressionType Property

Gets or sets the impression type of the NFRecord.

```
public NFImpressionType ImpressionType {get; set;}
```

Property value

One of the NFImpressionType values. The default is NFImpressionType.LiveScanPlain.

See Also

NFImpressionType

7.4.1.1.11. Minutiae Property

Gets minutiae collection.

```
public NFRecord.MinutiaCollection Minutiae {get;}
```

Property value

A NFRecord.MinutiaCollection that contains minutiae.

See Also

NFRecord.MinutiaCollection

7.4.1.1.12. MinutiaeNeighbours Property

Gets minutia neighbours format from NFRecord.

```
public MinutiaNeighboursCollection MinutiaeNeighbours {get;}
```

Property value

A MinutiaNeighboursCollection collection.

See Also

MinutiaNeighboursCollection

7.4.1.1.13. MinutiaFormat Property

Gets or sets minutia format minutia format from NFRecord.

```
public NFMinutiaFormat MinutiaFormat {get; set;}
```

Property value

One of the NFMinutiaFormat values. The default is NFMinutiaFormat.HasCurvature.

See Also

NFMinutiaFormat

7.4.1.1.14. PatternClass Property

Gets or sets pattern class.

```
public NFPatternClass PatternClass {get; set;}
```

Property value

One of the NFPatternClass values. By default is NFPatternClass.Unknown.

See Also

NFPatternClass

7.4.1.1.15. Position Property

Gets or sets finger position.

```
public NFPosition Position {get; set;}
```

Property value

One of the NFPosition values. By default is NFPosition.Unknown.

See Also

NFPosition

7.4.1.1.16. Quality Property

Gets or sets fingerprint quality.

```
public byte Quality {get; set;}
```

Property value

Fingerprint quality.

7.4.1.1.17. RidgeCountsType Property

Gets or sets ridge counts type.

```
public NFRidgeCountsType RidgeCountsType {get; set;}
```

Property value

One of the NFRidgeCountsType values.

See Also

NFRidgeCountsType

7.4.1.1.18. VertResolution Property

Gets vertical resolution of fingerprint image.

```
public ushort VertResolution {get;}
```

Property value

Vertical resolution in pixels per inch of fingerprint image.

See Also

HorzResolution

7.4.1.1.19. Width Property

Gets width of fingerprint image.

```
public ushort Width {get;}
```

Property value

Width of fingerprint image.

See Also

Height

7.4.1.1.20. Check Methods

Checks if format of packed `NFRecord` is correct.

7.4.1.1.20.1. Check(byte[] buffer)

Checks if format of packed `NFRecord` is correct.

```
public static void Check(
        byte[] buffer
);
```

Parameters

`buffer`	The packed `NFRecord` object.

See Also

Check

7.4.1.1.20.2. Check(IntPtr buffer, int bufferSize)

Checks if format of packed `NFRecord` is correct.

```
public static void Check(
        IntPtr buffer,
        int bufferSize
);
```

Parameters

`buffer`	Pointer to memory buffer that contains packed `NFRecord`.
`bufferSize`	Size of memory buffer that contains packed `NFRecord`.

See Also

Check

7.4.1.1.21. Clone Method

Creates `NFRecord` object from another `NFRecord` object.

```
public virtual object Clone();
```

Return Values

The new `NFRecord` object.

7.4.1.1.22. Dispose Method

Releases the resources used by NFRecord.

```
public void Dispose();
```

7.4.1.1.23. FromHandle Method

Creates `NFRecord` object from handle.

```
public static NFRecord FromHandle(
        IntPtr handle
);
```

Parameters

`handle`	Handle to unmanaged NFRecord object.

Return Values

`NFRecord` object.

See Also

NFRecord Constructors

7.4.1.1.24. GetCbeffProductType Method

```
public static ushort GetCbeffProductType(
    byte[] buffer
);
```

Parameters

`buffer`	The packed NFRecord object.

Return Values

The Cbeff product type.

See Also

CbeffProductType

7.4.1.1.25. GetG Method

Retrieves G from packed `NFRecord`.

```
public static byte GetG(
        byte[] buffer
);
```

Parameters

`buffer`	The packed `NFRecord` object.

Return Values

The G from packed `NFRecord`. G - average fingerprint ridge density.

See Also

G

7.4.1.1.26. GetHeight Method

Retrieves height of fingerprint image from packed NFRecord.

```
public static ushort GetHeight(
        byte[] buffer
);
```

Parameters

`buffer`	The packed `NFRecord` object.

Return Values

The height of fingerprint image.

See Also

GetWidth

7.4.1.1.27. GetHorzResolution Method

Retrieves horizontal resolution in pixels per inch of fingerprint image from packed `NFRecord`.

```
public static ushort GetHorzResolution(
        byte[] buffer
);
```

Parameters

`buffer`	The byte array of packed `NFRecord`.

Return Values

The horizontal resolution in pixels per inch of fingerprint image.

See Also

VertResolution

7.4.1.1.28. GetImpressionType Method

Retrieves impression type from packed NFRecord.

```
public static NFImpressionType GetImpressionType(
        byte[] buffer
);
```

Parameters

`buffer`	The byte array of packed `NFRecord`.

Return Values

A NFImpressionType enumeration member specifying impression type of fingerprint.

See Also

NFImpressionType | ImpressionType

7.4.1.1.29. GetMaxSize Method

Calculates the maximal `NFRecord` size.

7.4.1.1.29.1. GetMaxSize(NFMinutiaFormat,int,NFRidgeCountsType,int,int,int)

```
public static int GetMaxSize(
                NFMinutiaFormat minutiaFormat,
                int minutiaCount,
                NFRidgeCountsType ridgeCountsType,
                int coreCount,
                int deltaCount,
                int doubleCoreCount
        );
```

Parameters

`minutiaFormat`	One of the NFMinutiaFormat values.
`minutiaCount`	The minutiae count.
`ridgeCountsType`	One of the NFRidgeCountsType values.
`coreCount`	The cores count.
`deltaCount`	The deltas count.
`doubleCoreCount`	The double cores count.

Return Values

The maximal `NFRecord` size. The size depends on method parameters.

7.4.1.1.29.2. GetMaxSize(NFMinutiaFormat,int,NFRidgeCountsType,int,int,int,int,int)

```
public static int GetMaxSize(
                NFMinutiaFormat minutiaFormat,
                int minutiaCount,
                NFRidgeCountsType ridgeCountsType,
                int coreCount,
                int deltaCount,
                int doubleCoreCount,
                int boWidth,
                int boHeight
        );
```

Parameters

`minutiaFormat`	One of the NFMinutiaFormat values.
`minutiaCount`	The minutiae count.
`ridgeCountsType`	One of the NFRidgeCountsType values.
`coreCount`	The cores count.
`deltaCount`	The deltas count.
`doubleCoreCount`	The double cores count.
`boWidth`	For compatibility with VeriFinger.
`boHeight`	For compatibility with VeriFinger.

Return Values

The maximal `NFRecord` size. The size depends on method parameters.

7.4.1.1.30. GetMaxSizeV1 Method

Calculates the maximal version 1.0 `NFRecord` size.

7.4.1.1.30.1. GetMaxSizeV1(NFMinutiaFormat,int,int,int,int)

```
public static int GetMaxSizeV1(
        NFMinutiaFormat minutiaFormat,
        int minutiaCount,
        int coreCount,
        int deltaCount,
        int doubleCoreCount
);
```

Parameters

`minutiaFormat`	One of the NFMinutiaFormat values.
`minutiaCount`	The minutiae count.
`coreCount`	The cores count.
`deltaCount`	The deltas count.
`doubleCoreCount`	The double cores count.

Return Values

The maximal version 1.0 `NFRecord` size. The size depends on method parameters.

7.4.1.1.30.2. GetMaxSizeV1(NFMinutiaFormat,int,int,int,int,int,int)

```
public static int GetMaxSizeV1(
        NFMinutiaFormat minutiaFormat,
        int minutiaCount,
        int coreCount,
        int deltaCount,
        int doubleCoreCount,
        int boWidth,
        int boHeight
);
```

Parameters

`minutiaFormat`	One of the NFMinutiaFormat values.
`minutiaCount`	The minutiae count.
`coreCount`	The cores count.
`deltaCount`	The deltas count.
`doubleCoreCount`	The double cores count.
`boWidth`	For compatibility with VeriFinger.
`boHeight`	For compatibility with VeriFinger.

Return Values

The maximal version 1.0 `NFRecord` size. The size depends on method parameters.

7.4.1.1.31. GetPatternClass Method

Retrieves fingerprint pattern class from packed `NFRecord`.

```
public static NFPatternClass GetPatternClass(
        byte[] buffer
);
```

Parameters

`buffer`	The byte array of packed `NFRecord`.

Return Values

One of the NFPatternClass values.

See Also

PatternClass

7.4.1.1.32. GetPosition Method

Retrieves finger position from packed `NFRecord`.

```
public static NFPosition GetPosition(
        byte[] buffer
);
```

Parameters

`buffer`	The byte array of packed `NFRecord`.

Return Values

One of the NFPosition values.

See Also

NFPosition | Position

7.4.1.1.33. GetQuality Method

Retrieves fingerprint quality from packed `NFRecord`.

```
public static byte GetQuality(
        byte[] buffer
);
```

Parameters

`buffer`	The byte array of packed `NFRecord`.

Return Values

The value of fingerprint quality.

7.4.1.1.34. GetSize Methods

Calculates packed size of `NFRecord`.

7.4.1.1.34.1. GetSize()

Calculates packed size of NFRecord.

```
public int GetSize();
```

Return Values

The packed size of NFRecord.

Remarks

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

See Also

NFRecord Constructors | `GetSize(uint flags)`

7.4.1.1.34.2. GetSize(uint flags)

Calculates packed size of NFRecord. Behavior is controlled through flags.

```
public int GetSize(
        uint flags
);
```

Parameters

`flags`	Bitwise combination of zero or more flags

	that controls behavior of the method.

Return Values

The packed size of NFRecord.

Remarks

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

See Also

NFRecord Constructors | `GetSize()`

7.4.1.1.35. GetSizeV1 Method

Calculates version 1.0 packed size of NFRecord.

7.4.1.1.35.1. GetSizeV1()

Calculates version 1.0 packed size of NFRecord.

```
public int GetSizeV1()
```

Return Values

The packed size of NFRecord.

7.4.1.1.35.2. GetSizeV1(uint)

Calculates version 1.0 packed size of NFRecord. Behavior is controlled through flags.

```
public int GetSizeV1(
        uint flags
);
```

Parameters

`flags`	Bitwise combination of zero or more flags that controls behavior of the method.

Return Values

The packed size of NFRecord.

See Also

NFRecord Constructors | `GetSize`

7.4.1.1.36. GetVertResolution Method

Retrieves vertical resolution of fingerprint image from packed `NFRecord`.

```
public static ushort GetVertResolution(
        byte[] buffer
);
```

Parameters

`buffer`	The byte array of packed `NFRecord`.

Return Values

The vertical resolution in pixels per inch of fingerprint image.

See Also

GetHorzResolution

7.4.1.1.37. GetWidth Method

Retrieves width of fingerprint image from packed `NFRecord`.

```
public static ushort GetWidth(
        byte[] buffer
);
```

Parameters

`buffer`	The byte array of packed `NFRecord`.

Return Values

The width of fingerprint image.

See Also

GetHeight

7.4.1.1.38. Save Methods

Packs `NFRecord` to byte array.

7.4.1.1.38.1. Save()

Packs NFRecord to byte array.

```
public byte[] Save();
```

Return Values

The array of packed NFRecord.

Remarks

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

Note that blocked orientations are not packed. To pack them use Save with flags or SaveV1 method.

See Also

NFRecord Constructors

7.4.1.1.38.2. Save(uint flags)

Packs NFRecord to byte array. Behavior is controlled through flags.

```
public byte[] Save(
        uint flags
);
```

Parameters

`flags`	Bitwise combination of zero or more flags that controls behavior of the method.

Return Values

The byte array of packed NFRecord.

Remarks

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

Note that blocked orientations are not packed by default.

The following flags are supported:

See Also

NFRecord Constructors | Save | SaveV1

7.4.1.1.39. SaveV1 Method

Packs `NFRecord` to memory buffer in version 1.0 format.

7.4.1.1.39.1. SaveV1()

```
public byte[] SaveV1();
```

Return Values

The byte array of packed NFRecord.

See Also

Save

7.4.1.1.39.2. SaveV1(uint)

Packs NFRecord to memory buffer in version 1.0 format.

```
public byte[] SaveV1(
        uint flags
);
```

Parameters

`flags`	Bitwise combination of zero or more flags that controls behavior of the method.

Return Values

The byte array of packed NFRecord.

Remarks

Note that blocked orientations are not packed by default.

The following flags are supported: FlagSkipSingularPoints, FlagSaveBlockedOrients, FlagSkipCurvatures, FlagSkipGs.

See Also

NFRecord Constructors | Save

7.4.1.2. NFRecord.CoreCollection Class

Represents the collection of NFCore.

```
public sealed class CoreCollection: IList
```

Properties

Capacity	Gets or sets the number of elements that the CoreCollection can contain.
Count	Gets the number of items in the collection.
Item	Gets or sets the member from collection by index.

Methods

Add	Adds an object to the end of the CoreCollection.
Clear	Removes all elements from the CoreCollection.
`CopyTo`	Copies the CoreCollection or a portion of it to a onedimensional array.
`GetEnumerator`	Returns an enumerator that can be used to iterate through the CoreCollection.
Insert	Inserts an element into the CoreCollection at the specified index.
RemoveAt	Removes the element at the specified index within the `NFRecord.CoreCollection`.

7.4.1.2.1. Capacity Property

Gets or sets the number of elements that the CoreCollection can contain.

```
public int Capacity {get; set;}
```

Property value

The number of elements that the `NFRecord.CoreCollection` can contain.

See Also

Count

7.4.1.2.2. Count Property

Gets the number of items in the collection.

```
public int Count {get;}
```

Property value

The number of items in the collection.

7.4.1.2.3. NFRecord.CoreCollection.Item Property

Gets or sets the member from collection by index.

```
public NFCore this[
        int index
] {get; set;}
```

Parameters

index	The index of the NFCore structure to retrieve from the collection.

Property value

A NFCore structure.

See also

NFCore Count

7.4.1.2.4. Add Method

Adds an object to the end of the CoreCollection.

```
public int Add(
        NFCore value
);
```

Parameters

`value`	The NFCore to add to the collection.

Return Values

The zero-based index into the collection where the item was added.

See Also

Insert

7.4.1.2.5. Clear Method

Removes all elements from the CoreCollection.

```
public void Clear();
```

See Also

RemoveAt

7.4.1.2.6. CopyTo Method

Copies the CoreCollection or a portion of it to a onedimensional array.

```
public void CopyTo(Array array, int index)
```

Parameters

`array`	The one-dimensional array that is the destination of the elements copied from this collection.
`index`	The zero-based index in array at which copying begins.

7.4.1.2.7. GetEnumerator Method

Returns an enumerator that can be used to iterate through the CoreCollection.

```
public IEnumerator GetEnumerator();
```

Return Values

An IEnumerator that represents the checked index collection.

See Also

NFRecord.CoreCollection

7.4.1.2.8. Insert Method

Inserts an element into the CoreCollection at the specified index.

```
public void Insert(
        int index,
        NFCore value
);
```

Parameters

`index`	The zero-based index location where the NFCore is inserted.
`value`	The NFCore to add to the collection.

See Also

Add

7.4.1.2.9. RemoveAt Method

Removes the element at the specified index within the `NFRecord.CoreCollection`.

```
public void RemoveAt(
        int index
);
```

Parameters

`index`	The zero-based index of the NFCore to remove.

See Also

Clear

7.4.1.3. NFRecord.DeltaCollection Class

Represents the collection of NFDelta.

```
public sealed class DeltaCollection: IList
```

Properties

Capacity	Gets or sets the number of elements that the FmrFingerView.MinutiaCollection can contain.
Count	Gets the number of items in the collection.
Item	Gets or sets the member from collection by index.

Methods

Add	Adds an object to the `NFRecord.DeltaCollection`.
Clear	Removes all elements from the `NFRecord.DeltaCollection`.
`CopyTo`	Copies the DeltaCollection or a portion of it to a onedimensional array.
`GetEnumerator`	Returns an enumerator that can be used to iterate through the DeltaCollection.
Insert	Inserts an element into the `NFRecord.DeltaCollection` at the specified index.
RemoveAt	Removes the element at the specified index within the `NFRecord.DeltaCollection`.

7.4.1.3.1. Capacity Property

Gets or sets the number of elements that the FmrFingerView.MinutiaCollection can contain.

```
public int Capacity {get; set;}
```

Property value

The number of elements that the `NFRecord.DeltaCollection` can contain.

See Also

Count

7.4.1.3.2. Count Property

Gets the number of items in the collection.

```
public int Count {get;}
```

Property value

The number of items in the collection.

7.4.1.3.3. NFRecord.DeltaCollection.Item Property

Gets or sets the member from collection by index.

```
public NFDelta this[
        int index
] {get; set;}
```

Parameters

index	The index of the NFDelta structure to retrieve from the collection.

Property value

A NFDelta structure.

See also

Count

7.4.1.3.4. Add Method

Adds an object to the `NFRecord.DeltaCollection`.

```
public int Add(
        NFDelta value
);
```

Parameters

`value`	The NFDelta to add to the collection.

Return Values

The zero-based index into the collection where the item was added.

See Also

Insert

7.4.1.3.5. Clear Method

Removes all elements from the `NFRecord.DeltaCollection`.

```
public virtual void Clear();
```

See Also

RemoveAt

7.4.1.3.6. CopyTo Method

Copies the DeltaCollection or a portion of it to a onedimensional array.

```
public void CopyTo(Array array, int index)
```

Parameters

`array`	The one-dimensional array that is the destination of the elements copied from this collection.
`index`	The zero-based index in array at which copying begins.

7.4.1.3.7. GetEnumerator Method

Returns an enumerator that can be used to iterate through the DeltaCollection.

```
public IEnumerator GetEnumerator();
```

Return Values

An IEnumerator that represents the core collection.

7.4.1.3.8. Insert Method

Inserts an element into the `NFRecord.DeltaCollection` at the specified index.

```
public void Insert(
        int index,
        NFDelta value
);
```

Parameters

index	The zero-based index location where the NFDelta is inserted.
value	The NFDelta to add to the collection.

See Also

Add

7.4.1.3.9. RemoveAt Method

Removes the element at the specified index within the `NFRecord.DeltaCollection`.

```
public virtual void RemoveAt(
        int index
);
```

Parameters

index	The zero-based index of the NFDelta to remove.

See Also

Clear

7.4.1.4. NFRecord.DoubleCoreCollection Class

Represents the collection of NFDoubleCore.

```
public sealed class DoubleCoreCollection: IList
```

Properties

Capacity	Gets or sets the number of elements that the NFRecord.DoubleCoreCollection can contain.
Count	Gets the number of items in the collection.
Item	Gets or sets the member from collection by index.

Methods

Add	Adds an object to the `NFRecord.DoubleCoreCollection`.
Clear	Removes all elements from the `NFRecord.DoubleCoreCollection`.
`CopyTo`	Copies the DoubleCoreCollection or a portion of it to a onedimensional array.
`GetEnumerator`	Returns an enumerator that can be used to iterate through the DoubleCoreCollection.
Insert	Inserts an element into the `NFRecord.DoubleCoreCollection` at the specified index.
RemoveAt	Removes the element at the specified index of the `NFRecord.DoubleCoreCollection`.

7.4.1.4.1. Capacity Property

Gets or sets the number of elements that the NFRecord.DoubleCoreCollection can contain.

```
public int Capacity {get; set;}
```

Property value

The number of elements that the `NFRecord.DoubleCoreCollection` can contain.

See Also

Count

7.4.1.4.2. Count Property

Gets the number of items in the collection.

```
public int Count {get;}
```

Property value

The number of items in the collection.

7.4.1.4.3. Item Property

Gets or sets the member from collection by index.

```
public NFDoubleCore this[
        int index
        ] {get; set;}
```

Property value

A NFDoubleCore structure.

See also

NFDoubleCore Count

7.4.1.4.4. Add Method

Adds an object to the `NFRecord.DoubleCoreCollection`.

```
public int Add(
        NFDoubleCore value
);
```

Parameters

`value`	The NFDoubleCore to add to the collection.

Return Values

The zero-based index into the collection where the item was added.

See Also

Insert

7.4.1.4.5. Clear Method

Removes all elements from the `NFRecord.DoubleCoreCollection`.

```
public virtual void Clear();
```

See Also

RemoveAt

7.4.1.4.6. CopyTo Method

Copies the DoubleCoreCollection or a portion of it to a onedimensional array.

```
public void CopyTo(Array array, int index)
```

Parameters

array	The one-dimensional array that is the destination of the elements copied from this collection.
index	The zero-based index in array at which copying begins.

7.4.1.4.7. GetEnumerator Method

Returns an enumerator that can be used to iterate through the DoubleCollection.

```
public IEnumerator GetEnumerator();
```

Return Values

An IEnumerator that represents the double core collection.

7.4.1.4.8. Insert Method

Inserts an element into the `NFRecord.DoubleCoreCollection` at the specified index.

```
public void Insert(
        int index,
        NFDoubleCore value
);
```

Parameters

index	The zero-based index location where the NFDoubleCore is inserted.
value	The NFDoubleCore to add to the collection.

See Also

Add

7.4.1.4.9. RemoveAt Method

Removes the element at the specified index of the `NFRecord.DoubleCoreCollection`.

```
public virtual void RemoveAt(int index);
```

Parameters

`index`	The zero-based index of the NFDoubleCore to remove.

See Also

Clear

7.4.1.5. NFRecord.MinutiaCollection Class

Represents the collection of NFMinutia.

```
public sealed class MinutiaCollection: IList
```

Properties

Capacity	Gets or sets the number of elements that the NFRecord.MinutiaCollection can contain.
Count	Gets the number of items in the collection.
Item	Gets or sets the member from collection by index.

Methods

Add	Adds an object to the `NFRecord.MinutiaCollection`.
Clear	Removes all elements from the `NFRecord.MinutiaCollection`.
`CopyTo`	Copies the MinutiaCollection or a portion of it to a one- dimensional array.
`GetEnumerator`	Returns an enumerator that can be used to iterate through the MinutiaCollection.
Insert	Inserts an element into the `NFRecord.MinutiaCollection` at the specified index.
RemoveAt	Removes the element at the specified index

	within the `NFRecord.MinutiaCollection`.

7.4.1.5.1. Capacity Property

Gets or sets the number of elements that the NFRecord.MinutiaCollection can contain.

```
public int Capacity {get; set;}
```

Property value

The number of elements that the `NFRecord.MinutiaCollection` can contain.

See Also

Count

7.4.1.5.2. Count Property

Gets the number of items in the collection.

```
public int Count {get;}
```

Property value

The number of items in the collection.

7.4.1.5.3. MinutiaCollection.Item Property

Gets or sets the member from collection by index.

```
public NFMinutia this[
        int index
] {get; set;}
```

Parameters

index	The index of the NFMinutia structure to retrieve from the collection.

Property value

A NFMinutia structure.

See also

NFMinutia | Count

7.4.1.5.4. Add Method

Adds an object to the `NFRecord.MinutiaCollection`.

```
public int Add(
        NFMinutia value
);
```

Parameters

`value`	The NFMinutia to add to the collection.

Return Values

The zero-based index into the collection where the item was added.

See Also

Insert

7.4.1.5.5. Clear Method

Removes all elements from the `NFRecord.MinutiaCollection`.

```
public virtual void Clear()
```

See Also

RemoveAt

7.4.1.5.6. CopyTo Method

Copies the MinutiaCollection or a portion of it to a one- dimensional array.

```
public void CopyTo(Array array, int index)
```

Parameters

`array`	The one-dimensional array that is the destination of the elements copied from this collection.
`index`	The zero-based index in array at which copying begins.

7.4.1.5.7. GetEnumerator Method

Returns an enumerator that can be used to iterate through the MinutiaCollection.

```
public IEnumerator GetEnumerator();
```

Return Values

An IEnumerator that represents the minutia collection.

7.4.1.5.8. Insert Method

Inserts an element into the `NFRecord.MinutiaCollection` at the specified index.

```
public void Insert(
        int index,
        NFMinutia value
);
```

Parameters

`index`	The zero-based index location where the NFMinutia is inserted.
`value`	The NFMinutia to add to the collection.

See Also

Add

7.4.1.5.9. RemoveAt Method

Removes the element at the specified index within the `NFRecord.MinutiaCollection`.

```
public virtual void RemoveAt(
        int index
);
```

Parameters

`index`	The zero-based index of the NFMinutia to remove.

See Also

Clear

7.4.1.6. NFRecord.MinutiaNeighboursCollection Class

Represents the collection of NFMinutiaNeighbour.

Properties

Count	Gets or sets the number of elements that the MinutiaNeighboursCollection can contain.
Item	Gets the member of items in the collection.

Methods

GetCount	Retrieves the number of minutia neighbours at the specified index.
GetEnumerator	Returns an enumerator that can be used to iterate through the MinutiaNeighboursCollection.

7.4.1.6.1. Count Property

Gets or sets the number of elements that the MinutiaNeighboursCollection can contain.

```
public int Count {get;}
```

Property value

The number of items in the collection

7.4.1.6.2. NFRecord.MinutiaNeighboursCollection.Item Property

7.4.1.6.2.1. this[int]

Gets the members array from collection by index;

```
public NFMinutiaNeighbour[] this[
                int minutiaIndex
        ] {get;}
```

Parameters

minutiaIndex	The index of the specified minutia.

Property value

A NFMinutiaNeighbour structures array of specified minutia.

See Also

NFMinutiaNeighbour | Count

7.4.1.6.2.2. this[int, int]

Gets or sets the member from collection by index

```
public NFMinutiaNeighbour this[
                int minutiaIndex,
                int index
        ] {get; set;}
```

Parameters

minutiaIndex	The index of the specified minutia.
index	The index of the specified neighbour minutia to retrieve from the collection.

Property value

A NFMinutiaNeighbour structure of specified neighbour minutia.

See Also

NFMinutiaNeighbour | >Count

7.4.1.6.3. GetCount Method

Retrieves the number of minutia neighbours at the specified index.

```
public int GetCount(
        int minutiaIndex
);
```

Parameters

minutiaIndex	The index of the minutia.

Return Values

The number of the minutia neighbours.

7.4.1.6.4. GetEnumerator Method

Returns an enumerator that can be used to iterate through the MinutiaNeighboursCollection.

```
public IEnumerator GetEnumerator();
```

Return Values

An IEnumerator that represents the minutia neighbours collection.

7.4.1.7. NFCore Struct

The structure contains information of core.

```
public struct NFCore
```

Constructors

NFCore Constructor	Initializes a new instance of the `NFCore` structure.

Properties

Angle	Gets or sets Angle of core.
RawAngle	Gets or sets raw angle of core
X	Gets or sets y coordinate of core.
Y	Gets or sets x coordinate of core.

7.4.1.7.1. NFCore Constructor

7.4.1.7.1.1. NFCore(ushort, ushort)

Initializes a new instance of the NFCore structure.

```
public NFCore(
        ushort x,
        ushort y
);
```

Parameters

ushort	The x - coordinate of the core.
ushort	The y - coordinate of the core.

7.4.1.7.1.2. NFCore(ushort x, ushort y, int angle)

```
public NFCore(
        ushort x,
        ushort y,
        int angle
);;
```

Parameters

x	The x - coordinate of the core.
y	The y - coordinate of the core.
angle	The angle of the core.

7.4.1.7.1.3. NFCore(ushort x, ushort y, double angle)

```
public NFCore(
        ushort x,
        ushort y,
        double angle
);
```

Parameters

x	The x - coordinate of the core.
y	The y - coordinate of the core.
angle	The angle of the core.

7.4.1.7.2. Angle Property

Gets or sets Angle of core.

```
public double Angle {get; set;}
```

Property value

The angle of the core.

7.4.1.7.3. RawAngle Property

Gets or sets raw angle of core

```
public int RawAngle {get; set}
```

Property value

The raw angle of the core.

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.

7.4.1.7.4. X Property

Gets or sets y coordinate of core.

```
public int X {get; set;}
```

Property value

The y coordinate of the core.

Remarks

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

7.4.1.7.5. Y Property

Gets or sets x coordinate of core.

```
public int Y {get; set;}
```

Property value

The y coordinate of the core.

Remarks

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

7.4.1.8. NFDelta Struct

The structure contains information of delta.

```
public struct NFDelta
```

Constructors

NFDelta Constructor	Initializes a new instance of the NFDelta structure.

Properties

Angle1	Gets or sets the first angle of delta.
Angle2	Gets or sets the second angle of delta.
Angle3	Gets or sets the third angle of delta.
RawAngle1	Gets or sets the first row angle of the delta.
RawAngle2	Gets or sets the second row angle of the delta.
RawAngle3	Gets or sets the third row angle of the delta.
X	Gets or sets x coordinate of delta.
Y	Gets or sets y coordinate of delta.

7.4.1.8.1. NFDelta Constructor

Initializes a new instance of the NFDelta structure.

7.4.1.8.1.1. NFDelta(ushort x, ushort y)

```
public NFDelta(
        ushort x,
        ushort y
);
```

Parameters

x	The x - coordinate of the delta.
y	The y - coordinate of the delta.

7.4.1.8.1.2. NFDelta(ushort x, ushort y, int angle1, int angle2, int angle3)

```
public NFDelta(
        ushort x,
        ushort y,
        int angle1,
        int angle2,
        int angle3
);
```

Parameters

x	The x - coordinate of the delta.
y	The y - coordinate of the delta.
angle1	The first angle of the delta.
angle2	The second angle of the delta.
angle3	The third angle of the delta.

7.4.1.8.1.3. NFDelta(ushort x, ushort y, double angle1, double angle2, double angle3)

```
public NFDelta(
        ushort x,
        ushort y,
        double angle1,
        double angle2,
        double angle3
);
```

Parameters

x	The x - coordinate of the delta.
y	The y - coordinate of the delta.
angle1	The first angle of the delta.
angle2	The second angle of the delta.

angle3	The third angle of the delta.

7.4.1.8.2. Angle1 Property

Gets or sets the first angle of delta.

```
public double Angle1 {get; set;}
```

Property value

The first angle of the delta.

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.

7.4.1.8.3. Angle2 Property

Gets or sets the second angle of delta.

```
public double Angle2 {get; set;}
```

Property value

The second angle of the delta.

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.

7.4.1.8.4. Angle3 Property

Gets or sets the third angle of delta.

```
public double Angle3 {get; set;}
```

Property value

The third angle of the delta.

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.

7.4.1.8.5. RawAngle1 Property

Gets or sets the first row angle of the delta.

```
public int RawAngle1 {get; set;}
```

Property value

The first raw angle of the delta.

7.4.1.8.6. RawAngle2 Property

Gets or sets the second row angle of the delta.

```
public int RawAngle2 {get; set;}
```

Property value

The second raw angle of the delta.

7.4.1.8.7. RawAngle3 Property

Gets or sets the third row angle of the delta.

```
public int RawAngle3 {get; set;}
```

Property value

The third raw angle fog the delta.

7.4.1.8.8. X Property

Gets or sets x coordinate of delta.

```
public int X {get; set;}
```

Property value

The x coordinate of the delta.

Remarks

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

7.4.1.8.9. Y Property

Gets or sets y coordinate of delta.

```
public int Y {get; set;}
```

Property value

The x coordinate of the delta.

Remarks

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

7.4.1.9. NFDoubleCore Struct

The structure contains information of double core.

```
public struct NFDoubleCore
```

Constructors

NFDoubleCore Constructor	Initializes a new instance of the `NFDoubleCore` structure.

Properties

X	Gets or sets x coordinate of the double core.
Y	Gets or sets y coordinate of the double core.

7.4.1.9.1. NFDoubleCore Constructor

```
public NFDoubleCore(ushort x, ushort y);
```

Parameters

x	The x - coordinate of the double core.
y	The y - coordinate of the double core.

7.4.1.9.2. X Property

Gets or sets x coordinate of the double core.

```
public int X {get; set;}
```

Property value

The X coordinate of the double core.

7.4.1.9.3. Y Property

Gets or sets y coordinate of the double core.

```
public int Y {get; set;}
```

Property value

The Y coordinate of the double core.

7.4.1.10. NFMinutia Struct

The structure contains information of minutia.

```
public struct NFMinutia
```

Constructors

NFMinutia Constructor	Initializes a new instance of the NFMinutia structure.

Properties

Angle	Gets or sets the angle of the minutia.
Curvature	Gets or sets the ridge curvature near minutia.
G	Gets or sets the G (ridge density) near minu-

	tia.
Quality	Gets or sets quality of the minutia.
RawAngle	Gets or sets the raw angle of the minutia.
Type	Gets or sets the type of the minutia.
X	Gets or sets x coordinate of the minutia.
Y	Gets or sets y coordinate of the minutia.

7.4.1.10.1. NFMinutia Constructor

Initializes a new instance of the NFMinutia structure.

7.4.1.10.1.1. NFMinutia(ushort x, ushort y, NFMinutiaType type, byte angle)

```
public NFMinutia(
        ushort x,
        ushort y,
        NFMinutiaType type,
        byte angle
);
```

Parameters

x	The x - coordinate of the minutia.
y	The y - coordinate of the minutia.
type	One of the NFMinutiaType values.
angle	The angle of the minutia.

7.4.1.10.1.2. NFMinutia(ushort x, ushort y, NFMinutiaType type, double angle)

```
public NFMinutia(
        ushort x,
        ushort y,
        NFMinutiaType type,
        double angle
);
```

Parameters

x	The x - coordinate of the minutia.
y	The y - coordinate of the minutia.
type	One of the NFMinutiaType values.
angle	The angle of the minutia.

7.4.1.10.1.3. NFMinutia(ushort x, ushort y, NFMinutiaType type, byte angle, byte quality, byte curvature, byte g)

```
public NFMinutia(
        ushort x,
        ushort y,
        NFMinutiaType type,
        byte angle,
        byte quality,
        byte curvature,
        byte g
);
```

Parameters

x	The x - coordinate of the minutia.
y	The y - coordinate of the minutia.
type	One of the NFMinutiaType values.
angle	The angle of the minutia.
quality	The quality of the minutia.
curvature	The ridge curvature near minutia.
g	The G (ridge density) near minutia.

7.4.1.10.1.4. NFMinutia(ushort x, ushort y, NFMinutiaType type, double angle, byte quality, byte curvature, byte g)

```
public NFMinutia(
        ushort x,
        ushort y,
        NFMinutiaType type,
        double angle,
        byte quality,
        byte curvature,
        byte g
);
```

Parameters

x	The x - coordinate of the minutia.
y	The y - coordinate of the minutia.
type	One of the NFMinutiaType values.
angle	The angle of the minutia.
quality	The quality of the minutia.
curvature	The ridge curvature near minutia.
g	The G (ridge density) near minutia.

7.4.1.10.2. Angle Property

Gets or sets the angle of the minutia.

```
public double Angle {get; set;}
```

Property value

The angle of the minutia.

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.

7.4.1.10.3. Curvature Property

Gets or sets the ridge curvature near minutia.

```
public byte Curvature {get; set;}
```

Property value

The ridge curvature near minutia.

Remarks

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

7.4.1.10.4. G Property

Gets or sets the G (ridge density) near minutia.

```
public byte G {get; set;}
```

Property value

The G (ridge density) near minutia.

Remarks

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

7.4.1.10.5. Quality Property

Gets or sets quality of the minutia.

```
public byte Quality {get; set;}
```

Property value

The quality of the minutia.

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.

7.4.1.10.6. RawAngle Property

Gets or sets the raw angle of the minutia.

```
public byte RawAngle {get; set;}
```

Property value

The raw angle of the minutia.

7.4.1.10.7. Type Property

Gets or sets the type of the minutia.

```
public NFMinutiaType Type {get; set;}
```

Property value

One of the NFMinutiaType values.

7.4.1.10.8. X Property

Gets or sets x coordinate of the minutia.

```
public int X {get; set;}
```

Property value

The X coordinate of the minutia.

Remarks

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

7.4.1.10.9. Y Property

Gets or sets y coordinate of the minutia.

```
public int Y {get; set;}
```

Property value

The Y coordinate of the minutia.

Remarks

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

7.4.1.11. NFMinutiaNeighbour Struct

The structure contains information of minutia neighbour.

```
public struct NFMinutiaNeighbour
```

Constructors

NFMinutiaNeighbour Constructor	Initializes a new instance of the `NFMinutiaNeighbour` structure.

Fields

Empty	Represents a minutia neighbour that is null reference.

Properties

Index	Gets or sets the index of minutia neighbour.
RidgeCount	Gets or sets the ridge count between the minutia and minutia neighbour.

7.4.1.11.1. NFMinutiaNeighbour Constructor

```
public NFMinutiaNeighbour(
        int index,
        byte ridgeCount
);
```

Parameters

`index`	The index of neighbour.
`ridgeCount`	The ridge count of neighbour.

7.4.1.11.2. Empty Field

Represents a NFMinutiaNeighbour that is a null reference.

```
public static readonly NFMinutiaNeighbour Empty;
```

7.4.1.11.3. Index Property

Gets or sets the index of minutia neighbour.

```
public int Index {get; set;}
```

Property value

The index of minutia neighbour.

7.4.1.11.4. RidgeCount Property

Gets or sets the ridge count between the minutia and minutia neighbour.

```
public int RidgeCount {get; set;}
```

Property value

The ridge count between the minutia and minutia neighbour.

7.4.1.12. NFImpressionType Enumeration

Specifies the impression types.

```
public enum NFImpressionType
```

Members

Member name	**Description**
LatentImpression	Latent impression fingerprint.
LatentLift	Latent lift fingerprint.
LatentPhoto	Latent photo fingerprint.
LatentTracing	Latent tracing fingerprint.
LiveScanContactless	Live-scanned fingerprint using contactless device.
LiveScanPlain	Live-scanned plain fingerprint.
LiveScanRolled	Live-scanned rolled fingerprint.
NonliveScanPlain	Nonlive-scanned (from paper) plain fingerprint.
NonliveScanRolled	Nonlive-scanned (from paper) rolled fingerprint.
Swipe	Live-scanned fingerprint by sliding the finger across a "swipe" sensor.

7.4.1.13. NFMinutiaFormat Enumeration

Specifies formats of minutia. This enumeration allows a bitwise combination of its member values.

```
public enum NFMinutiaFormat
```

Members

Member name	Description
HasCurvature	If near minutia is ridge curvature
HasG	If near minutia is G(ridge density).
HasQuality	The quality of the minutia.

7.4.1.14. NFMinutiaType Enumeration

Specifies types of minutia.

```
public enum NFMinutiaType
```

Members

Member name	Description
Bifurcation	The minutia that is a bifurcation of a ridge.
End	The minutia that is an end of a ridge.
Unknown	The type of the minutia is unknown.

7.4.1.15. NFPatternClass Enumeration

Specifies pattern class of the fingerprint.

```
public enum NFPatternClass
```

Members

Member name	Description
AccidentalWhorl	Accidental whorl pattern class.
Amputation	Amputation. Pattern class is not available.
CentralPocketLoop	Central pocket loop pattern class.
DoubleLoop	Double loop pattern class.
LeftSlantLoop	Left slant loop pattern class.
PlainArch	Plain arch pattern class.

Member name	Description
PlainWhorl	Plain whorl pattern class.
RadialLoop	Radial loop pattern class.
RightSlantLoop	Right slant loop pattern class.
Scar	Scar. Pattern class is not available.
TentedArch	Tented arch pattern class.
UlnarLoop	Ulnar loop pattern class.
Unknown	Unknown pattern class.
Whorl	Whorl pattern class.

Remarks

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

7.4.1.16. NFPosition Enumeration

Specifies finger position.

```
public enum NFPosition
```

Members

Member name	Description
LeftIndex	Index finger of the left hand.
LeftLittle	Little finger of the left hand.
LeftMiddle	Middle finger of the left hand.
LeftRing	Ring finger of the left hand.
LeftThumb	Thumb of the left hand.
RightThumb	Thumb of the right hand.
RightIndex	Index finger of the right hand.
RightLittle	Little finger of the right hand.
RightMiddle	Middle finger of the right hand.

Member name	Description
RightRing	Ring finger of the right hand.
Unknown	Unknown finger.

7.4.1.17. NFRidgeCountsType Enumeration

Specifies type of ridge counts contained in NFRecord.

```
public enum NFRidgeCountsType
```

Members

Member name	Description
EightNeighbours	The NFRecord contains ridge counts to closest minutia in each of the eight sectors of each minutia. First sector starts at minutia angle.
EightNeighboursWithIndexes	The NFRecord contains ridge counts to eight neighbours of each minutia.
FourNeighbours	The NFRecord contains ridge counts to closest minutia in each of the four sectors of each minutia. First sector starts at minutia angle.
FourNeighboursWithIndexes	The NFRecord contains ridge counts to four neighbours of each minutia.
None	The NFRecord does not contain ridge counts.
Unspecified	For internal use.

7.5. Neurotec.Biometrics.VFExtractor Library

Provides functionality for extracting Neurotechnologija Finger Records from fingerprint images using VeriFinger algorithm.

DLL: `Neurotec.Biometrics.VFExtractor.dll`.

Namespaces

Neurotec.Biometrics	Provides functionality for extracting Neurotechnologija Finger Records from fingerprint images using VeriFinger algorithm.

7.5.1. Neurotec.Biometrics Namespace

Provides functionality for extracting Neurotechnologija Finger Records from fingerprint images using VeriFinger algorithm.

Classes

Class	**Description**
VFExtractor	Provides methods for extracting Neurotechnologija Finger Records (NFRecords) from fingerprint images using VeriFinger algorithm encapsulated in Neurotechnologija Fingerprint Features Extractor VF (VFExtractor) object.

Enumerations

Enumeration	**Description**
VfeReturnedImage	Specifies the returned image.
VfeTemplateSize	

7.5.1.1. VFExtractor Class

Wrapper for VeriFinger SDK VFExtractor module is implemented in `Neurotec.Biometrics.VFExtractor.dll`.

Constructors

VFExtractor Constructor	Initializes a new instance of the `VFExtractor` class.

Properties

GeneralizationMaximalRotation	Gets or sets maximal rotation of two features collection to each other.

GeneralizationThreshold	Gets or sets generalization threshold.
IsRegistered	Gets a value indicating whether VFExtractor library has been registered.
Mode	Gets or sets scanners mode.
ReturnedImage	Gets or sets the image type.
TemplateSize	Gets or sets template size. The value can be one of VfeTemplateSize enumeration members.

Methods

CopyParameters	Copies the parameters values from one VFExtractor object to another.
Dispose	Releases the resources used by VFExtractor.
Extract	Extracts features from a fingerprint image.
ExtractUnpacked	Extracts features from a fingerprint image and do not packs.
Generalize	Generalizes count features collections to single features collection.
GeneralizeUnpacked	Unpackes and generalizes count features collections to single features collection.
GetMaxTemplateSize	Retrieves maximal size of packed NFRecord the specified VFExtractor can extract.
GetParameter	Retrieves parameter by parameter identifier.
GetStaticParameter	Retrieves static parameter by parameter identifier.
Reset	Resets all parameters of VFExtractor object to default values.
SetParameter	Sets the parameter by parameter identifier.
SetStaticParameter	Sets the static parameter by parameter identifier.

Constants

DllName	Name of DLL containing unmanaged part of this class.
ParameterMode	Identifier specifying mode (parameter value set) parameter of type uint. Parameter value can be one of the ModeXXX.
ModeGeneral ModeDigitalPersonaUareU ModeBiometrikaFX2000 ModeBiometrikaFX3000 ModeKeytronicSecureDesktop ModeIdentixTouchView ModeIdentixDfr2090 ModePreciseBiometrics100CS ModeUpekTouchChip ModeIdenticatorTechnologyDF90 ModeAuthentecAFS2 ModeAuthentecAes4000 ModeAuthentecAes2501B ModeAtmelFingerchip ModeBmfBlp100 ModeSecugenHamster ModeEthentica ModeCrossmatchVerifier300 ModeNitgenFingkeyHamster ModeTestechBioI ModeDigentIzzix ModeStartekFM200 ModeFujitsuMBF200 ModeFutronicFS80 ModeLighTuningLttC500 ModeTacomaCmos	Represents scanners.
ParameterCopyright	Identifier specifying library copyright static read-only parameter of type string.
ParameterGeneralizationMaximalRotation	Maximal rotation of two features collection to each other. Must be in range 0°..180°.
ParameterGeneralizationThreshold	Has the same meaning for features generalization as ParameterMatchingThreshold parameter for features matching.
ParameterName	Identifier specifying library name static read-only parameter of type string.
ParameterQualityThreshold	Identifier specifies image quality threshold.
ParameterReturnedImage	Identifier specifying kind of image returned

	after extraction parameter of type int. Parameter value can be one of the VfeReturnedImage enumeration members.
ParameterTemplateSize	Identifier specifying template size parameter. Parameter value can be one of the VfeTemplateSize enumeration members.
ParameterUseQuality	
ParameterVersionHigh	Identifier specifying high part of library version static read-only parameter of type uint. Two high-order bytes of parameter value specify major version and two low-order bytes - minor version.
ParameterVersionLow	Identifier specifying low part of library version static read-only parameter of type uint. Two high-order bytes of parameter value specify major (build) version and two low-order bytes - minor (release) version.

7.5.1.1.1. VFExtractor Constructor

Initializes a new instance of the VFExtractor class.

```
public VFExtractor();
```

See Also

`VFExtractor` Class

7.5.1.1.2. GeneralizationMaximalRotation Property

Gets or sets maximal rotation of two features collection to each other.

```
public byte GeneralizationMaximalRotation {get; set;}
```

Property value

See Also

7.5.1.1.3. GeneralizationThreshold Property

Gets or sets generalization threshold.

```
public int GeneralizationThreshold {get; set;}
```

Property value

See Also

7.5.1.1.4. IsRegistered Property

Gets a value indicating whether VFExtractor library has been registered.

```
public static bool IsRegistered {get}
```

Property value

`true` if VFExtractor library is registered; otherwise, `false`.

7.5.1.1.5. Mode Property

Gets or sets scanners mode.

```
public uint Mode {get, set}
```

Property value

One of the Mode constants.

See Also

Mode

7.5.1.1.6. ReturnedImage Property

Gets or sets the image type.

```
public VfeReturnedImage ReturnedImage {get, set};
```

Property value

One of the VfeReturnedImage value.

See Also

VfeReturnedImage

7.5.1.1.7. TemplateSize Property

Gets or sets template size. The value can be one of VfeTemplateSize enumeration members.

```
public VfeTemplateSize TemplateSize {get; set;}
```

Property value

See Also

7.5.1.1.8. CopyParameters Method

Copies the parameters values from one VFExtractor object to another.

```
public static void CopyParameters(
        VFExtractor sourceExtractor,
        VFExtractor destinationExtractor
);
```

Parameters

`sourceExtractor`	Object of the `VFExtractor` class.
`destinationExtractor`	Object of the `VFExtractor` class.

See Also

Constants

7.5.1.1.9. Dispose Method

Releases the resources used by VFExtractor.

```
public void Dispose();
```

7.5.1.1.10. Extract Methods

Extracts features from a fingerprint image.

7.5.1.1.10.1. int Extract(ushort, ushort, uint, ushort, ushort, IntPtr,NFPosition, NFImpressionType, IntPtr, int, out bool)

```
public int Extract(
        ushort width,
        ushort height,
        uint stride,
        ushort horzResolution,
        ushort vertResolution,
        IntPtr pixels,
        NFPosition position,
        NFImpressionType impressionType,
        IntPtr buffer,
        int bufferSize,
        out bool isLowQuality
);
```

Parameters

width	The fingerprint image width.
height	The fingerprint image height.
stride	The fingerprint image width, plus the width of the alignment bytes.
horzResolution	Horizontal resolution in pixels per inch of fingerprint image.
vertResolution	Vertical resolution in pixels per inch of fingerprint image.
pixels	The pointer to array of fingerprint image.
position	The finger NFPosition.
impressionType	The finger NFImpressionType.
buffer	Pointer to memory buffer that contains packed NFRecord.
bufferSize	Size of memory buffer that contains packed NFRecord.
isLowQuality	`true` if fingerprint image is low quality; otherwise, `false`.

Return Values

Size of extracted NFRecord.

Remarks

position and *impressionType* are written to extracted NFRecord.

See Also

Extract

7.5.1.1.10.2. int Extract(ushort, ushort, uint, ushort, ushort, IntPtr,NFPosition, NFImpressionType, byte[], out bool)

```
public int Extract(
        ushort width,
        ushort height,
        uint stride,
        ushort horzResolution,
        ushort vertResolution,
        IntPtr pixels,
        NFPosition position,
        NFImpressionType impressionType,
        byte[] buffer,
        out bool isLowQuality
);
```

Parameters

`width`	The fingerprint image width.
`height`	The fingerprint image height.
`stride`	The fingerprint image width, plus the width of the alignment bytes.
`horzResolution`	Horizontal resolution in pixels per inch of fingerprint image.
`vertResolution`	Vertical resolution in pixels per inch of fingerprint image.
`pixels`	The pointer to array of fingerprint image.
`position`	The finger NFPosition.
`impressionType`	The finger NFImpressionType.
`buffer`	Byte array that contains packed NFRecord.
`isLowQuality`	`true` if fingerprint image is low quality; otherwise, `false`.

Return Values

Size of extracted NFRecord.

Remarks

`position` and *`impressionType`* are written to extracted NFRecord.

See Also

Extract

7.5.1.1.10.3. byte[] Extract(ushort, ushort, uint, ushort, ushort, IntPtr,NFPosition, NFImpressionType, out bool)

Extracts finger features from fingerprint image.

```
public byte[] Extract(
        ushort width,
        ushort height,
        uint stride,
        ushort horzResolution,
        ushort vertResolution,
        IntPtr pixels,
        NFPosition position,
        NFImpressionType impressionType,
        out bool isLowQuality
        )
```

Parameters

width	The fingerprint image width.
height	The fingerprint image height.
stride	The fingerprint image width, plus the width of the alignment bytes.
horzResolution	Horizontal resolution in pixels per inch of fingerprint image.
vertResolution	Vertical resolution in pixels per inch of fingerprint image.
pixels	The pointer to array of fingerprint image.
position	The finger NFPosition.
impressionType	The finger NFImpressionType.
buffer	Byte array that contains packed NFRecord.
isLowQuality	`true` if fingerprint image is low quality; otherwise, `false`.

Return Values

The array of packed NFRecord.

Remarks

`position` and `impressionType` are written to extracted NFRecord.

See Also

Extract

7.5.1.1.10.4. int Extract(NGrayscaleImage, NFPosition, NFImpressionType, IntPtr, int, out bool)

```
public int Extract(
        NGrayscaleImage image,
        NFPosition position,
        NFImpressionType impressionType,
        IntPtr buffer,
        int bufferSize,
        out bool isLowQuality
);
```

Parameters

`image`	The NGrayscaleImage image.
`position`	The finger NFPosition.
`impressionType`	The finger NFImpressionType.
`buffer`	Byte array that contains packed NFRecord.
`bufferSize`	Size of memory buffer that contains packed NFRecord.
`isLowQuality`	`true` if fingerprint image is low quality; otherwise, `false`.

Return Values

Size of extracted NFRecord.

Remarks

`position` and `impressionType` are written to extracted NFRecord.

See Also

Extract

7.5.1.1.10.5. int Extract(NGrayscaleImage, NFPosition, NFImpressionType, byte[], out bool)

```
public int Extract(
        NGrayscaleImage image,
        NFPosition position,
        NFImpressionType impressionType,
        byte[] buffer,
        out bool isLowQuality
);
```

Parameters

`image`	The NGrayscaleImage image.
`position`	The finger NFPosition.
`impressionType`	The finger NFImpressionType.
`buffer`	Byte array that contains packed NFRecord.
`isLowQuality`	`true` if fingerprint image is low quality; otherwise, `false`.

Return Values

Size of extracted NFRecord.

Remarks

`position` and *`impressionType`* are written to extracted NFRecord.

See Also

Extract

7.5.1.1.10.6. byte[] Extract(NGrayscaleImage, NFPosition, NFImpressionType, out bool)

Extracts finger features from NGrayscaleImage object.

```
public byte[] Extract(
        NGrayscaleImage image,
        NFPosition position,
        NFImpressionType impressionType,
        bool isLowQuality
        );
```

Parameters

`image`	The NGrayscaleImage image.
`position`	The finger NFPosition.
`impressionType`	The finger NFImpressionType.
`isLowQuality`	`true` if fingerprint image is low quality; otherwise, `false`.

Return Values

The array of packed NFRecord.

Remarks

`position` and *`impressionType`* are written to extracted NFRecord.

See Also

Extract

7.5.1.1.11. ExtractUnpacked Method

Extracts features from a fingerprint image and do not packs.

7.5.1.1.11.1. ExtractUnpacked(NGrayscaleImage,NFPosition,NFImpressionType,out bool)

```
public NFRecord ExtractUnpacked(
        NGrayscaleImage image,
        NFPosition position,
        NFImpressionType impressionType,
        out bool isLowQuality
);
```

Parameters

`image`	The NGrayscaleImage image.
`position`	The finger NFPosition.
`impressionType`	The finger NFImpressionType.
`isLowQuality`	`true` if fingerprint image is low quality; otherwise, `false`.

Return Values

The NFRecord object.

Remarks

`position` and `impressionType` are written to extracted NFRecord.

See Also

NFRecord

7.5.1.1.11.2. ExtractUnpacked(ushort,ushort,uint,ushort,ushort,IntPtr,NFPosition,NFImpressionType,out bool)

```
public NFRecord ExtractUnpacked(
        ushort width,
        ushort height,
        uint stride,
        ushort horzResolution,
        ushort vertResolution,
        IntPtr pixels,
        NFPosition position,
        NFImpressionType impressionType,
        out bool isLowQuality
);
```

Parameters

width	The fingerprint image width.
height	The fingerprint image height.
stride	The fingerprint image width, plus the width of the alignment bytes.
horzResolution	Horizontal resolution in pixels per inch of fingerprint image.
vertResolution	Vertical resolution in pixels per inch of fingerprint image.
pixels	The pointer to array of fingerprint image.
position	The finger NFPosition.
impressionType	The finger NFImpressionType.
isLowQuality	`true` if fingerprint image is low quality; otherwise, `false`.

Return Values

The NFRecord object.

Remarks

`position` and *`impressionType`* are written to extracted NFRecord.

See Also

NFRecord

7.5.1.1.12. Generalize Methods

Generalizes count features collections to single features collection.

7.5.1.1.12.1. int Generalize(IntPtr[], int[], IntPtr, int, out int)

```
public int Generalize(
        IntPtr[] templates,
        int[] templateSizes,
        IntPtr buffer,
        int bufferSize,
        out int baseTemplateIndex
);
```

Parameters

`templates`	Pointer to NFRecord objects array.
`templateSizes`	An array that contains NFRecords sizes.
`buffer`	Pointer to memory buffer that contains NFRecord object.
`bufferSize`	Size of memory buffer that contains NFRecord.
`baseTemplateIndex`	Index of main generalization template.

Return Values

Size of generalized NFRecord.

Remarks

If templates can not generalized, method through `bufferSize` returns zero and through `buffer` - null.

See Also

NFRecord | Extract

7.5.1.1.12.2. int Generalize(byte[][], IntPtr, int, out int)

```
public int Generalize(
        byte[][] templates,
        IntPtr buffer,
        int bufferSize,
        out int baseTemplateIndex
);
```

Parameters

templates	A byte array of NFRecord objects byte arrays.
buffer	Pointer to memory buffer that contains NFRecord object.
bufferSize	Size of memory buffer that contains NFRecord.
baseTemplateIndex	Index of main generalization template.

Return Values

Size of generalized NFRecord.

See Also

NFRecord | Extract

7.5.1.1.12.3. int Generalize(byte[][], byte[], out int)

```
public int Generalize(
        byte[][] templates,
        byte[] buffer,
        out int baseTemplateIndex
);
```

Parameters

templates	A byte array of NFRecord objects byte arrays.
buffer	Byte array that contains generalized NFRecord.
baseTemplateIndex	Index of main generalization template.

Return Values

Size of generalized NFRecord.

See Also

NFRecord | Extract

7.5.1.1.12.4. byte[] Generalize(byte[][], out int)

```
public byte[] Generalize(
        byte[][] templates,
        out int baseTemplateIndex
);
```

Parameters

templates	A byte array of NFRecord objects byte arrays.
baseTemplateIndex	Index of main generalization template.

Return Values

Byte array that contains generalized NFRecord.

See Also

NFRecord | Extract

7.5.1.1.13. GeneralizeUnpacked Methods

Unpackes and generalizes count features collections to single features collection.

7.5.1.1.13.1. GeneralizeUnpacked(byte[][], out int)

Unpackes and generalizes count features collections to single features collection.

```
public NFRecord GeneralizeUnpacked(
    byte[][] templates,
    out int baseTemplateIndex
);
```

Parameters

templates	A byte array that contains NFRecord objects bytes arrays.
baseTemplateIndex	Index of main generalization template.

Return Values

The generalized NFRecord object.

See Also

Extract

7.5.1.1.13.2. GeneralizeUnpacked(IntPtr[], int[], out int)

Unpackes and generalizes count features collections to single features collection.

```
public NFRecord GeneralizeUnpacked(
    IntPtr[] templates,
    int[] templateSizes,
    out int baseTemplateIndex
);
```

Parameters

templates	Pointer to NFRecord objects array.
templateSizes	An array that contains NFRecord objects sizes.
baseTemplateIndex	Index of main generalization template.

Return Values

The generalized NFRecord object.

See Also

Extract

7.5.1.1.14. GetMaxTemplateSize Method

Retrieves maximal size of packed NFRecord the specified VFExtractor can extract.

```
public int GetMaxTemplateSize();
```

Return Values

Returns maximal template size.

7.5.1.1.15. GetParameter Method

Retrieves parameter by parameter identifier.

```
public object GetParameter(
        ushort parameterId
);
```

Parameters

`parameterId`	The parameter identifier.

Return Values

The parameter value.

Remarks

The following values can be used for *`parameterId`*:

- ParameterMode
- ParameterReturnedImage

See Also

SetParameter | CopyParameters

7.5.1.1.16. GetStaticParameter Method

Retrieves static parameter by parameter identifier.

```
public static object GetStaticParameter(
        ushort parameterId
);
```

Parameters

`parameterId`	The parameter identifier.

Return Values

The parameter value.

Remarks

The following values can be used for *`parameterId`*:

- ParameterCopyright
- ParameterName
- ParameterVersionHigh
- ParameterVersionLow

See Also

SetStaticParameter | CopyParameters

7.5.1.1.17. Reset Method

Resets all parameters of VFExtractor object to default values.

```
public void Reset();
```

7.5.1.1.18. SetParameter Method

Sets the parameter by parameter identifier.

```
public void SetParameter(
        ushort parameterId,
        object value
);
```

Parameters

`parameterId`	The parameter identifier.

`value`	The parameter value.

Remarks

The following values can be used for *`parameterId`*:

- ParameterMode
- ParameterReturnedImage

See Also

GetParameter | CopyParameters

7.5.1.1.19. SetStaticParameter Method

Sets the static parameter by parameter identifier.

```
public static void SetStaticParameter(
        ushort parameterId,
        object value
);
```

Parameters

`parameterId`	The parameter identifier.
`value`	The parameter object.

Remarks

Unused method.

See Also

GetStaticParameter | CopyParameters

7.5.1.2. VfeReturnedImage Enumeration

Specifies the returned image.

```
public enum VfeReturnedImage
```

Members

Member name	Description
None	No image.
Binarized	The binarized image.
Skeletonized	The skeletonized image.

7.5.1.3. VfeTemplateSize Enumeration

```
public enum VfeTemplateSize
```

Members

Member name	Description
Large	
Small	

7.6. Neurotec.Biometrics.VFMatcher Library

Provides functionality for comparing Neurotechnologija Finger Records using VeriFinger algorithm.

DLL: `Neurotec.Biometrics.VFMatcher.dll`.

Namespaces

Neurotec.Biometrics	Provides functionality for comparing Neurotechnologija Finger Records using VeriFinger algorithm.

7.6.1. Neurotec.Biometrics Namespace

Provides functionality for comparing Neurotechnologija Finger Records using VeriFinger algorithm.

Classes

VfmMatchDetails	Specifies the matching information.
VFMatcher	Provides functionality for comparing Neuro-

	technologija Finger Records (NFRecords) using VeriFinger algorithm encapsulated in Neurotechnologija Finger Matcher VF (VFMatcher) object.

Enumerations

Enumeration	Description
VfmSpeed	

7.6.1.1. VfmMatchDetails Class

Specifies the matching information.

Properties

CenterX	X rotation point coordinate of the second (Verify or IdentifyNext) matched template.
CenterY	Y rotation point coordinate of the second (Verify or IdentifyNext) matched template.
MatedMinutiae	The array of structure NIndexPair.
RawRotation	The rotation of fingerprint image.
Rotation	The rotation of fingerprint image in radians.
Score	Specifies score of two matched fingerprints.
TranslationX	The x translation of fingerprint image.
TranslationY	The y translation of fingerprint image.

Methods

Dispose	Releases the resources used by VfmMatchDetails.

7.6.1.1.1. CenterX Property

X rotation point coordinate of the second (Verify or IdentifyNext) matched template.

```
public int CenterX{get};
```

7.6.1.1.2. CenterY Property

Y rotation point coordinate of the second (Verify or IdentifyNext) matched template.

```
public int CenterY{get};
```

7.6.1.1.3. MatedMinutiae Property

The array of structure NIndexPair.

```
public NIndexPair[] MatedMinutiae {get;}
```

Property value

The array of NIndexPair structures.

See Also

NIndexPair

7.6.1.1.4. RawRotation Property

The rotation of fingerprint image.

```
public byte RawRotation {get;}
```

7.6.1.1.5. Rotation Property

The rotation of fingerprint image in radians.

```
public double Rotation {get;}
```

7.6.1.1.6. Score Property

Specifies score of two matched fingerprints.

```
public int Score {get;}
```

Property value

The score of two matched fingerprints.

7.6.1.1.7. TranslationX Property

The x translation of fingerprint image.

```
public int TranslationX {get;}
```

7.6.1.1.8. TranslationY Property

The y translation of fingerprint image.

```
public int TranslationY {get;}
```

7.6.1.1.9. Dispose Method

Releases the resources used by VfmMatchDetails.

```
public void Dispose();
```

7.6.1.2. VFMatcher Class

Provides functionality for comparing Neurotechnologija Finger Records (NFRecords) using VeriFinger algorithm encapsulated in Neurotechnologija Finger Matcher VF (VFMatcher) object.

Constructors

VFMatcher Constructor	Initializes a new instance of the `VFMatcher` class.

Properties

IsRegistered	Gets a value indicating whether Neurotec.Biometrics.VFMatcher library is registered.
MatchingThreshold	Gets or sets the matching threshold.
MaximalRotation	Gets or sets maximal rotation.
Mode	Gets or sets scanners mode.

Methods

CopyParameters	Copies parameters values from one VFMatcher object to another VFMatcher object.

Dispose	Releases the resources used by VFMatcher.
GetParameter	Gets value of parameter by parameter id.
GetStaticParameter	Gets value of static parameter by parameter id.
IdentifyEnd	Ends identification process.
IdentifyNext	Compares the specified NFRecord with one identification was started with.
IdentifyStart	Starts identification with the specified NFRecord.
Reset	Resets all VFMatcher parameters to default values.
SetParameter	Sets value of parameter by parameter id.
SetStaticParameter	Sets value of static parameter by parameter id.
Verify	Compars two NFRecords.

Constants

DllName	Name of DLL containing unmanaged part of this class.
ModeGeneral ModeDigitalPersonaUareU ModeBiometrikaFX2000 ModeBiometrikaFX3000 ModeKeytronicSecureDesktop ModeIdentixTouchView ModeIdentixDfr2090 ModePreciseBiometrics100CS ModeUpekTouchChip ModeIdenticatorTechnologyDF90 ModeAuthentecAFS2 ModeAuthentecAes4000 ModeAuthentecAes2501B ModeAtmelFingerchip ModeBmfBlp100 ModeSecugenHamster ModeEthentica ModeCrossmatchVerifier300	Represents scanners.

ModeNitgenFingkeyHamster ModeTestechBioI ModeDigentIzzix ModeStartekFM200 ModeFujitsuMBF200 ModeFutronicFS80 ModeLighTuningLttC500 ModeTacomaCmos	
ParameterCopyright	Identifier specifying library copyright static read-only parameter of type string.
ParameterMatchingThreshold	Identifier specifying matching threshold (biggest allowed FAR) parameter of type int. Parameter value is equal to - 12 * log10(FAR) and must be not less than zero (for example, 48 for FAR = 0.01%).
ParameterMaximalRotation	Identifier specifying modulus of maximal rotation allowed between two matched NFRecords parameter of type byte. Parameter value is specified in 180/128 degrees units and can not be greater than 128 (+-180 degrees).
ParameterMode	Identifier specifying mode (parameter value set) parameter of type uint. Parameter value can be one of the ModeXXX.
ParameterName	Identifier specifying library name static read-only parameter of type string.
ParameterVersionHigh	Identifier specifying high part of library version static read-only parameter of type uint. Two high-order bytes of parameter value specify major version and two low-order bytes - minor version.
ParameterVersionLow	Identifier specifying low part of library version static read-only parameter of type uint. Two high-order bytes of parameter value specify major (build) version and two low-order bytes - minor (release) version.

7.6.1.2.1. VFMatcher Constructor

Initializes a new instance of the VFMatcher class.

```
public VFMatcher();
```

7.6.1.2.2. IsRegistered Property

Gets a value indicating whether Neurotec.Biometrics.VFMatcher library is registered.

```
public static bool IsRegistered {get;}
```

Property value

`true` if Neurotec.Biometrics.VFMatcher library is registered; otherwise, `false`.

7.6.1.2.3. MatchingThreshold Property

Gets or sets the matching threshold.

```
public int MatchingThreshold {get; set;}
```

Property value

The matching threshold.

See Also

ParameterMatchingThreshold

7.6.1.2.4. MaximalRotation Property

Gets or sets maximal rotation.

```
public byte MaximalRotation {get; set;}
```

Property value

The maximal rotation.

See Also

ParameterMaximalRotation

7.6.1.2.5. Mode Property

```
public uint Mode {get; set;}
```

Property value

The scanner type.

See Also

Constants

7.6.1.2.6. CopyParameters Method

Copies parameters values from one VFMatcher object to another VFMatcher object.

```
public static void CopyParameters(
        VFMatcher sourceMatcher,
        VFMatcher destinationMatcher
);
```

Parameters

sourceMatcher	The VFMatcher object.
destinationMatcher	The VFMatcher object.

See Also

Constants

7.6.1.2.7. Dispose Method

Releases the resources used by VFMatcher.

```
public void Dispose();
```

7.6.1.2.8. GetParameter Method

Gets value of parameter by parameter id.

```
public object GetParameter(
        ushort parameterId
);
```

Parameters

parameterId	The parameter identifier.

Return Values

The parameter value.

Remarks

The following values can be used for *`parameterId`*:

- ParameterMatchingThreshold
- ParameterMaximalRotation
- ParameterMode

See Also

SetParameter | CopyParameters

7.6.1.2.9. GetStaticParameter Method

Gets value of static parameter by parameter id.

```
public static object GetStaticParameter(
        ushort parameterId
);
```

Parameters

`parameterId`	The parameter identifier.

Return Values

The parameter value.

Return Values

The parameter value.

Remarks

The following values can be used for *`parameterId`*:

- ParameterCopyright
- ParameterName
- ParameterVersionHigh
- ParameterVersionLow

See Also

SetStaticParameter | CopyParameters

7.6.1.2.10. IdentifyEnd Method

Ends identification process.

```
public void IdentifyEnd();
```

Remarks

See Also

IdentifyStart | IdentifyNext

7.6.1.2.11. IdentifyNext Method

Compares the specified NFRecord with one identification was started with.

7.6.1.2.11.1. IdentifyNext(IntPtr,int)

```
public int IdentifyNext(
        IntPtr template,
        int templateSize
);
```

Parameters

`template`	Pointer to the memory block that contains packed NFRecord.
`templateSize`	The template size.

Return Values

The matching score.

Remarks

Method returns zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see ParameterMatchingThreshold and SetParameter method), and is greater than or equal to matching threshold otherwise.

See Also

IdentifyStart | IdentifyEnd | NFRecord

7.6.1.2.11.2. IdentifyNext(IntPtr,int,VfmMatchDetails)

```
public int IdentifyNext(
        IntPtr template,
        int templateSize,
        VfmMatchDetails matchDetails
);
```

Parameters

template	Pointer to the memory block that contains packed NFRecord.
templateSize	The template size.
matchDetails	The VfmMatchDetails object.

Return Values

The matching score.

Remarks

Method returns zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see ParameterMatchingThreshold and SetParameter method), and is greater than or equal to matching threshold otherwise.

See Also

IdentifyStart | IdentifyEnd | VfmMatchDetails | NFRecord

7.6.1.2.11.3. IdentifyNext(byte[])

```
public int IdentifyNext(
        byte[] template
);
```

Parameters

template	The byte array with packed NFRecord.

Return Values

The matching score.

Remarks

Method returns zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see ParameterMatchingThreshold and SetParameter method), and is greater than or equal to matching threshold otherwise.

See Also

IdentifyStart | IdentifyEnd | NFRecord

7.6.1.2.11.4. IdentifyNext(byte[],VfmMatchDetails)

```
public int IdentifyNext(
        byte[] template,
        VfmMatchDetails matchDetails
);
```

Parameters

`template`	The byte array with packed NFRecord.
`matchDetails`	The VfmMatchDetails object.

Return Values

The matching score.

Remarks

Method returns zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see ParameterMatchingThreshold and SetParameter method), and is greater than or equal to matching threshold otherwise.

See Also

IdentifyStart | IdentifyEnd | VfmMatchDetails | NFRecord

7.6.1.2.12. IdentifyStart Method

Starts identification with the specified NFRecord.

7.6.1.2.12.1. IdentifyStart(IntPtr, int templateSize)

```
public void IdentifyStart(
        IntPtr template,
        int templateSize
);
```

Parameters

template	Pointer to the memory block that contains packed NFRecord.
templateSize	The template size.

See Also

IdentifyNext | IdentifyEnd | NFRecord

7.6.1.2.12.2. IdentifyStart(IntPtr,int,out VfmMatchDetails)

```
public void IdentifyStart(
        IntPtr template,
        int templateSize,
        out VfmMatchDetails matchDetails
);
```

Parameters

template	Pointer to the memory block that contains packed NFRecord.
templateSize	The template size.
matchDetails	The VfmMatchDetails object.

See Also

IdentifyNext | IdentifyEnd | VfmMatchDetails | NFRecord

7.6.1.2.12.3. IdentifyStart(byte[])

```
public void IdentifyStart(
        byte[] template
);
```

Parameters

template	The byte array with packed NFRecord.

See Also

IdentifyNext | IdentifyEnd | NFRecord

7.6.1.2.12.4. IdentifyStart(byte[],out VfmMatchDetails)

```
public void IdentifyStart(
        byte[] template,
        out VfmMatchDetails matchDetails
);
```

Parameters

`template`	The byte array with packed NFRecord.
`matchDetails`	The VfmMatchDetails object.

See Also

IdentifyNext | IdentifyEnd | VfmMatchDetails | NFRecord

7.6.1.2.13. Reset Method

Resets all VFMatcher parameters to default values.

```
public void Reset();
```

7.6.1.2.14. SetParameter Method

Sets value of parameter by parameter id.

```
public void SetParameter(
        ushort parameterId,
        object value
);
```

Parameters

`parameterId`	The parameter identifier.
`value`	The parameter value.

Remarks

The following values can be used for *`parameterId`*:

- ParameterMatchingThreshold
- ParameterMaximalRotation

- ParameterMode

See Also

GetParameter | CopyParameters

7.6.1.2.15. SetStaticParameter Method

Sets value of static parameter by parameter id.

```
public static void SetStaticParameter(
        ushort parameterId,
        object value
);
```

Parameters

parameterId	The parameter identifier.
value	The parameter value.

Remarks

Unused mothod.

See Also

GetStaticParameter | CopyParameters

7.6.1.2.16. Verify Method

Compars two NFRecords.

7.6.1.2.16.1. Verify(IntPtr,int,IntPtr,int)

```
public int Verify(
        IntPtr template1,
        int template1Size,
        IntPtr template2,
        int template2Size
);
```

Parameters

template1	Pointer to the memory block that contains packed NFRecord.

template1Size	The template size.
template2	Pointer to the memory block that contains packed NFRecord.
template2Size	The template size.

Return Values

The matching score.

Remarks

Method returns zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see ParameterMatchingThreshold and SetParameter method), and is greater than or equal to matching threshold otherwise.

See Also

NFRecord

7.6.1.2.16.2. Verify(IntPtr,int,IntPtr,int,out VfmMatchDetails)

```
public int Verify(
        IntPtr template1,
        int template1Size,
        IntPtr template2,
        int template2Size,
        out VfmMatchDetails matchDetails
);
```

Parameters

template1	Pointer to the memory block that contains packed NFRecord.
template1Size	The template size.
template2	Pointer to the memory block that contains packed NFRecord.
template2Size	The template size.
matchDetails	The VfmMatchDetails object.

Return Values

The matching score.

Remarks

Method returns zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see ParameterMatchingThreshold and SetParameter method), and is greater than or equal to matching threshold otherwise.

See Also

VfmMatchDetails | NFRecord

7.6.1.2.16.3. Verify(byte[],byte[])

```
public int Verify(
        byte[] template1,
        byte[] template2
);
```

Parameters

template1	The byte array with packed NFRecord.
template2	The byte array with packed NFRecord.

Return Values

The matching score.

Remarks

Method returns zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see ParameterMatchingThreshold and SetParameter method), and is greater than or equal to matching threshold otherwise.

See Also

NFRecord

7.6.1.2.16.4. Verify(byte[],byte[],out VfmMatchDetails)

```
public int Verify(
        byte[] template1,
        byte[] template2,
        out VfmMatchDetails matchDetails
);
```

Parameters

`template1`	The byte array with packed NFRecord.
`template2`	The byte array with packed NFRecord.
`matchDetails`	The VfmMatchDetails object.

Return Values

The matching score.

Remarks

Method returns zero if similarity is less than matching threshold, i.e. two NFRecords do not match (see ParameterMatchingThreshold and SetParameter method), and is greater than or equal to matching threshold otherwise.

See Also

VfmMatchDetails | NFRecord

7.6.1.3. VfmSpeed Enumeration

```
public enum VfmSpeed
```

Members

Member name	**Description**
High	
Low	

7.7. Neurotec.Images Library

Provides classes that enable loading, saving and converting images in various formats.

DLL: `Neurotec.Images.dll`.

Namespaces

Neurotec.Images	Contains classes that enable loading, saving and converting images in various formats.

7.7.1. Neurotec.Images Namespace

Contains classes that enable loading, saving and converting images in various formats.

Classes

Bmp	Provides functionality for loading and saving images in BMP format.
NGrayscaleImage	Provides functionality for managing 8-bit grayscale images.
NImage	Provides functionality for managing images.
NImageFormat	Provides functionality for loading and saving images in format-neutral style.
NImageFormat.ImageFormatCollection	Represents the collection of formats in a NImageFormat.
NImages	Provides library registration and other additional functionality.
NMonochromeImage	Provides functionality for managing 1-bit monochrome images.
NRgbImage	Provides functionality for managing 24-bit RGB images.
Tiff	Provides functionality for loading and saving images in TIFF format.

Structures

NPixelFormat	Provides functionality for work with pixel format.
NRgb	Represents an RGB color.

7.7.1.1. Bmp Class

Provides functionality for loading and saving images in BMP format.

Methods

LoadImage	Creates a new instance of the `NImage` class.

LoadImageFromBitmap	Creates a new instance of the `NImage` class from `Bitmap`.
LoadImageFromHBitmap	Creates a new instance of the `NImage` class from Windows HBITMAP.
SaveImage	Saves `NImage`.
SaveImageToBitmap	Saves `NImage` to `Bitmap`.
SaveImageToHBitmap	Saves `NImage` to Windows HBITMAP.

7.7.1.1.1. LoadImage Method.

Creates a new instance of the `NImage` class.

7.7.1.1.1.1. LoadImage (string)

Creates a new instance of the `NImage` class from file.

```
public static NImage LoadImage(
        string fileName
);
```

Parameters

`fileName`	A string that contains the name of the file.

Return Values

A NImage object.

See Also

NImage | LoadImage | SaveImage

7.7.1.1.1.2. LoadImage (IntPtr, int)

Creates a new instance of the `NImage` class from memory buffer.

```
public static NImage LoadImage(
        IntPtr buffer,
        int bufferLength
);
```

Parameters

`buffer`	Pointer to memory buffer.
`bufferLength`	Size of memory buffer.

Return Values

A NImage object.

See Also

NImage | LoadImage | SaveImage

7.7.1.1.1.3. LoadImage (byte[])

Creates a new instance of the `NImage` class from byte array.

```
public static NImage LoadImage(
        byte[] buffer
);
```

Parameters

`buffer`	A byte array.

Return Values

A NImage object.

See Also

NImage | LoadImage | SaveImage

7.7.1.1.2. LoadImageFromBitmap Method

Creates a new instance of the `NImage` class from `Bitmap`.

7.7.1.1.2.1. LoadImageFromBitmap (Bitmap)

Creates a new instance of the `NImage` class from `Bitmap`.

```
public static NImage LoadImageFromBitmap(
        Bitmap bitmap
);
```

Parameters

`bitmap`	A `Bitmap` class object.

Return Values

A NImage object.

See Also

NImage | SaveImageToBitmap

7.7.1.1.2.2. LoadImageFromBitmap (Bitmap, float, float)

Creates a new instance of the `NImage` class from `Bitmap` with specified resolution.

```
public static NImage LoadImageFromBitmap(
        Bitmap bitmap,
        float horzResolution,
        float vertResolution
);
```

Parameters

`bitmap`	A `Bitmap` class object.
`horzResolution`	A horizontal resolution in pixels per inch of fingerprint image.
`vertResolution`	A vertical resolution in pixels per inch of fingerprint image.

Return Values

A NImage object.

See Also

NImage | SaveImageToBitmap

7.7.1.1.3. LoadImageFromHBitmap Method

Creates a new instance of the `NImage` class from Windows HBITMAP.

```
public static NImage LoadImageFromHBitmap(
        IntPtr hBitmap
);
```

Parameters

`hBitmap`	Pointer to handle that specifies Windows HBITMAP.

Return Values

A NImage object.

See Also

NImage | SaveImageToHBitmap

7.7.1.1.4. SaveImage Method

Saves `NImage`.

7.7.1.1.4.1. void SaveImage (NImage, string)

Saves `NImage` to file.

```
public static void SaveImage(
        NImage image,
        string fileName
);
```

Parameters

`image`	A NImage object.
`fileName`	A string that contains the name of the file.

See Also

NImage | LoadImage

7.7.1.1.4.2. void SaveImage (NImage, Stream)

Saves `NImage` to stream.

```
public static void SaveImage(
        NImage image,
        Stream stream
);
```

Parameters

`image`	A NImage object.
`stream`	The data stream used to save the image.

See Also

NImage | LoadImage

7.7.1.1.4.3. byte[] SaveImage (NImage)

Saves `NImage` to byte array.

```
public static byte[] SaveImage(
        NImage image
);
```

Parameters

`image`	A NImage object.

Return Values

A byte array.

See Also

NImage | LoadImage

7.7.1.1.5. SaveImageToBitmap Method

Saves `NImage` to `Bitmap`.

```
public static Bitmap SaveImageToBitmap(
        NImage image
);
```

Parameters

`image`	A NImage object.

Return Values

A `Bitmap` object.

See Also

NImage | LoadImageFromBitmap

7.7.1.1.6. SaveImageToHBitmap Method

Saves `NImage` to Windows HBITMAP.

```
public static IntPtr SaveImageToHBitmap(
        NImage image
);
```

Parameters

`image`	A NImage object.

Return Values

A pointer to handle of Windows HBITMAP.

See Also

NImage | LoadImageFromHBitmap

7.7.1.2. NGrayscaleImage Class

Provides functionality for managing 8-bit grayscale images.

Properties

Item	Gets or sets the color of the specified pixel in NImage object.

7.7.1.2.1. NGrayscaleImage.Item Property

Gets or sets the color of the specified pixel in NImage object.

```
public const byte this[
        uint x,
        uint y
] {get; set;}
```

Parameters

`x`	The x coordinate of the pixel.

y	The y coordinate of the pixel.

Property value

A color of specified pixel.

7.7.1.3. NImage Class

Provides functionality for managing images.

Properties

Handle	Gets handle to unmanaged NImage object.
Height	Gets height of fingerprint image from NImage object.
HorzResolution	Gets horizontal resolution in pixels per inch of fingerprint image.
LongSize	Gets size of NImage object.
LongStride	Gets stride of fingerprint image from NImage object.
PixelFormat	Gets NPixelFormat of NImage object.
Pixels	Gets pointer to array of pixels from NImage object.
Size	Gets size of NImage object.
Stride	Gets stride of fingerprint image from NImage object.
VertResolution	Gets vertical resolution in pixels per inch of fingerprint image.
Width	Gets width of fingerprint image from NImage object.

Methods

Clone	Creates NImage object from another NImage object.
Create	Creates an empty NImage object.

Dispose	Releases the resources used by NImage.
FromBitmap	Creates `NImage` from Bitmap.
FromData	Creates `NImage` object from data.
FromFile	Creates `NImage` object from file.
FromHandle	Creates `NImage` object from handle.
FromHBitmap	Creates a new instance of the NImage class from Windows HBITMAP.
FromImage	Creates `NImage` object from another `NImage` object.
GetWrapper	Creates `NImage` object wrapper.
Save	Saves `NImage` object to file.
ToBitmap	Creates a `Bitmap`.
ToHBitmap	Creates Windows HBITMAP.

7.7.1.3.1. Handle Property

Gets handle to unmanaged NImage object.

```
public IntPtr Handle {get;}
```

Property value

A handle to unmanaged NImage object.

See Also

NImage

7.7.1.3.2. Height Property

Gets height of fingerprint image from NImage object.

```
public uint Height {get;}
```

Property value

A height of fingerprint image.

See Also

NImage Width | Stride

7.7.1.3.3. HorzResolution Property

Gets horizontal resolution in pixels per inch of fingerprint image.

```
public float HorzResolution {get;}
```

Property value

A horizontal resolution in pixels per inch of fingerprint image.

Remarks

Horizontal resolution equal to zero means that it is not applicable for the image.

See Also

NImage VertResolution

7.7.1.3.4. LongSize Property

Gets size of NImage object.

```
public ulong LongSize {get;}
```

Property value

A size of NImage object.

Remarks

Size of memory block containing image pixels is equal to image height multiplied by image stride. For more information see Height and Stride properties.

See Also

NImage | Height | Stride

7.7.1.3.5. LongStride Property

Gets stride of fingerprint image from NImage object.

```
public ulong LongStride {get;}
```

Property value

A stride of fingerprint image.

Stride (size of one row) of the image depends on image pixel format and width. It can not be less than value obtained with GetRowLongSize or GetRowSize methods with arguments obtained with PixelFormat and Width properties.

See Also

NImage

7.7.1.3.6. PixelFormat Property

Gets NPixelFormat of NImage object.

```
public NPixelFormat PixelFormat {get;}
```

Property value

A NPixelFormat structure.

See Also

NImage

7.7.1.3.7. Pixels Property

Gets pointer to array of pixels from NImage object.

```
public IntPtr Pixels {get;}
```

Property value

A pointer to pixel array.

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 PixelFormat, Width, Height, Stride, and Size properties.

See Also

NImage | PixelFormat | Width | Height | Stride | Size

7.7.1.3.8. Size Property

Gets size of NImage object.

```
public uint Size {get;}
```

Property value

A size of NImage object.

Remarks

Size of memory block containing image pixels is equal to image height multiplied by image stride. For more information see Height and Stride properties.

See Also

NImage | Height | Stride

7.7.1.3.9. Stride Property

Gets stride of fingerprint image from NImage object.

```
public uint Stride {get;}
```

Property value

A stride of fingerprint image.

Stride (size of one row) of the image depends on image pixel format and width. It can not be less than value obtained with GetRowLongSize or GetRowSize methods with arguments obtained with PixelFormat and Width properties.

See Also

NImage

7.7.1.3.10. VertResolution Property

Gets vertical resolution in pixels per inch of fingerprint image.

```
public float VertResolution {get;}
```

Property value

A vertical resolution in pixels per inch of fingerprint image.

Remarks

Vertical resolution equal to zero means that it is not applicable for the image.

See Also

NImage | HorzResolution

7.7.1.3.11. Width Property

Gets width of fingerprint image from NImage object.

```
public uint Width {get;}
```

Property value

A width of fingerprint image.

See Also

NImage Height | Stride

7.7.1.3.12. Clone Method

Creates NImage object from another NImage object.

```
public object Clone();
```

Return Values

A NImage object.

See Also

NImage

7.7.1.3.13. Create Method

Creates an empty NImage object.

7.7.1.3.13.1. Create (NPixelFormat, uint, uint, uint, float, float)

Creates an empty NImage object.

```
public static NImage Create(
        NPixelFormat pixelFormat,
        uint width,
        uint height,
        uint stride,
        float horzResolution,
        float vertResolution
);
```

Parameters

pixelFormat	A NPixelFormat of fingerprint image.
width	A width of fingerprint image.
height	A height of fingerprint image.
stride	A stride of fingerprint image.
horzResolution	A horizontal resolution in pixels per inch of fingerprint image.
vertResolution	A vertical resolution in pixels per inch of fingerprint image.

Return Values

A NImage object.

If *stride* is zero then image stride is automatically calculated. For more information on image stride see Stride method.

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

See Also

NImage

7.7.1.3.13.2. Create (NPixelFormat, uint, uint, ulong, float, float)

Creates an empty NImage object.

```
public static NImage Create(
        NPixelFormat pixelFormat,
        uint width,
        uint height,
        ulong stride,
        float horzResolution,
        float vertResolution
);
```

Parameters

pixelFormat	A NPixelFormat of fingerprint image.
width	A width of fingerprint image.
height	A height of fingerprint image.
stride	A stride of fingerprint image.

`horzResolution`	A horizontal resolution in pixels per inch of fingerprint image.
`vertResolution`	A vertical resolution in pixels per inch of fingerprint image.

Return Values

A NImage object.

If *`stride`* is zero then image stride is automatically calculated. For more information on image stride see Stride method.

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

See Also

NImage

7.7.1.3.14. Dispose Method

Releases the resources used by NImage.

```
public void Dispose();
```

See Also

NImage

7.7.1.3.15. FromBitmap Method

Creates `NImage` from Bitmap.

```
public static NImage FromBitmap(
        Bitmap bitmap
);
```

Parameters

`bitmap`	An object used to work with images defined by pixel data.

Return Values

A NImage object.

See Also

NImage

7.7.1.3.16. FromData Method

Creates `NImage` object from data.

7.7.1.3.16.1. FromData (NPixelFormat, uint, uint, uint, float, float, uint, IntPtr)

Creates `NImage` object from data with specified resolution.

```
public static NImage FromData(
        NPixelFormat pixelFormat,
        uint width,
        uint height,
        uint stride,
        float horzResolution,
        float vertResolution,
        uint srcStride,
        IntPtr srcPixels
);
```

Parameters

pixelFormat	A NPixelFormat of fingerprint image.
width	A width of fingerprint image.
height	A height of fingerprint image.
stride	A stride of fingerprint image.
horzResolution	A horizontal resolution in pixels per inch of fingerprint image.
vertResolution	A vertical resolution in pixels per inch of fingerprint image.
srcStride	A stride of source fingerprint image.
srcPixels	A pointer to source pixel array.

Return Values

A NImage object.

If *`stride`* is zero then image stride is automatically calculated. For more information on image stride see Stride property.

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

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

See Also

NImage

7.7.1.3.16.2. FromData (NPixelFormat, uint, uint, ulong, float, float, ulong, IntPtr)

Creates `NImage` object from data with specified resolution.

```
public static NImage FromData(
        NPixelFormat pixelFormat,
        uint width,
        uint height,
        ulong stride,
        float horzResolution,
        float vertResolution,
        ulong srcStride,
        IntPtr srcPixels
);
```

Parameters

`pixelFormat`	A NPixelFormat of fingerprint image.
`width`	A width of fingerprint image.
`height`	A height of fingerprint image.
`stride`	A stride of fingerprint image.
`horzResolution`	A horizontal resolution in pixels per inch of fingerprint image.
`vertResolution`	A vertical resolution in pixels per inch of fingerprint image.
`srcStride`	A stride of source fingerprint image.
`srcPixels`	A pointer to source pixel array.

Return Values

A NImage object.

See Also

NImage

7.7.1.3.17. FromFile Method

Creates `NImage` object from file.

7.7.1.3.17.1. FromFile (string)

Creates `NImage` object from file.

```
public static NImage FromFile(
        string fileName
);
```

Parameters

fileName	A string that contains the name of the file.

Return Values

A NImage object.

See Also

NImage

7.7.1.3.17.2. FromFile (string, NImageFormat)

Creates `NImage` object from file with specified NImageFormat.

```
public static NImage FromFile(
        string fileName,
        NPixelFormat imageFormat
);
```

Parameters

fileName	A string that contains the name of the file.
imageFormat	An image NImageFormat object.

Return Values

A NImage object.

See Also

NImage | NImageFormat

7.7.1.3.18. FromHandle Method

Creates `NImage` object from handle.

```
public static NImage FromHandle(
        IntPtr handle
);
```

Parameters

`handle`	A pointer to handle.

Return Values

A NImage object.

See Also

NImage | Handle

7.7.1.3.19. FromHBitmap Method

Creates a new instance of the NImage class from Windows HBITMAP.

```
public static NImage FromHBitmap(
        IntPtr hBitmap
);
```

Parameters

`hBitmap`	Pointer to handle that specifies Windows HBITMAP.

Return Values

A NImage object.

See Also

NImage

7.7.1.3.20. FromImage Method

Creates `NImage` object from another `NImage` object.

7.7.1.3.20.1. FromImage (NPixelFormat, uint, NImage)

Creates `NImage` object from another `NImage` object.

```
public static NImage FromImage(
        NPixelFormat pixelFormat,
        uint stride,
        NImage srcImage
);
```

Parameters

`pixelFormat`	A NPixelFormat of fingerprint image.
`stride`	A stride of fingerprint image.
`srcImage`	A `NImage` source object.

Return Values

A NImage object.

Remarks

If *`stride`* is zero then image stride is automatically calculated. For more information on image stride see Stride property.

See Also

NImage | NPixelFormat

7.7.1.3.20.2. FromImage (NPixelFormat, ulong, NImage)

Creates `NImage` object from another `NImage` object.

```
public static NImage FromImage(
        NPixelFormat pixelFormat,
        ulong stride,
        NImage srcImage
);
```

Parameters

`pixelFormat`	A NPixelFormat of fingerprint image.
`stride`	A stride of fingerprint image.

`srcImage`	A `NImage` source object.

Return Values

A NImage object.

See Also

NImage | NPixelFormat

7.7.1.3.20.3. FromImage (NPixelFormat, uint, float, float, NImage)

Creates `NImage` object from another `NImage` object with specified resolution.

```
public static NImage FromImage(
        NPixelFormat pixelFormat,
        uint stride,
        float horzResolution,
        float vertResolution,
        NImage srcImage
);
```

Parameters

`pixelFormat`	A NPixelFormat of fingerprint image.
`stride`	A stride of fingerprint image.
`horzResolution`	A horizontal resolution in pixels per inch of fingerprint image.
`vertResolution`	A vertical resolution in pixels per inch of fingerprint image.
`srcImage`	A `NImage` source object.

Return Values

A NImage object.

Remarks

If *`stride`* is zero then image stride is automatically calculated. For more information on image stride see Stride property.

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

See Also

NImage | NPixelFormat

7.7.1.3.20.4. FromImage (NPixelFormat, ulong, float, float, NImage)

Creates `NImage` object from another `NImage` object with specified resolution.

```
public static NImage FromImage(
        NPixelFormat pixelFormat,
        ulong stride,
        float horzResolution,
        float vertResolution,
        NImage srcImage
);
```

Parameters

pixelFormat	A NPixelFormat of fingerprint image.
stride	A stride of fingerprint image.
horzResolution	A horizontal resolution in pixels per inch of fingerprint image.
vertResolution	A vertical resolution in pixels per inch of fingerprint image.
srcImage	A `NImage` source object.

Return Values

A NImage object.

See Also

NImage | NPixelFormat

7.7.1.3.21. GetWrapper Method

Creates `NImage` object wrapper.

7.7.1.3.21.1. GetWrapper (NPixelFormat, uint, uint, uint, float, float, IntPtr, bool)

Creates `NImage` object wrapper.

```
public static NImage GetWrapper(
        NPixelFormat pixelFormat,
        uint width,
        uint height,
        uint stride,
        float horzResolution,
        float vertResolution,
        IntPtr pixels,
        bool ownsPixels
);
```

Parameters

pixelFormat	A NPixelFormat of fingerprint image.
width	A width of fingerprint image.
height	A height of fingerprint image.
stride	A stride of fingerprint image.
horzResolution	A horizontal resolution in pixels per inch of fingerprint image.
vertResolution	A vertical resolution in pixels per inch of fingerprint image.
pixels	Pointer to memory block containing pixels for the image.
ownsPixels	Specifies whether pixels will be automatically deleted with the image (if set to true).

Return Values

A NImage object.

Remarks

For more information on image stride see Stride property.

Format of memory block pixels points to must be the same as described in Pixels property.

pixels must not be deleted during lifetime of the image. If *ownsPixels* is `true` 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 | NPixelFormat | Pixels

7.7.1.3.21.2. GetWrapper (NPixelFormat, uint, uint, ulong, float, float, IntPtr,

bool)

Creates `NImage` object wrapper.

```
public static NImage GetWrapper(
        NPixelFormat pixelFormat,
        uint width,
        uint height,
        ulong stride,
        float horzResolution,
        float vertResolution,
        IntPtr pixels,
        bool ownsPixels
);
```

Parameters

pixelFormat	A NPixelFormat of fingerprint image.
width	A width of fingerprint image.
height	A height of fingerprint image.
stride	A stride of fingerprint image.
horzResolution	A horizontal resolution in pixels per inch of fingerprint image.
vertResolution	A vertical resolution in pixels per inch of fingerprint image.
pixels	Pointer to memory block containing pixels for the image.
ownsPixels	Specifies whether pixels will be automatically deleted with the image (if set to true).

Return Values

A NImage object.

Remarks

Format of memory block pixels points to must be the same as described in Pixels property.

See Also

NImage | NPixelFormat | Pixels

7.7.1.3.22. Save Method

Saves `NImage` object to file.

7.7.1.3.22.1. Save (string)

Saves `NImage` object to file.

```
public void Save(
        string fileName
);
```

Parameters

`fileName`	A string that contains the name of the file.

See Also

NImage

7.7.1.3.22.2. Save (string, NImageFormat)

Saves `NImage` object to file with specified NImageFormat.

```
public void Save(
        string fileName,
        NImageFormat imageFormat
);
```

Parameters

`fileName`	A string that contains the name of the file.
`imageFormat`	An image NImageFormat object.

See Also

NImage | NImageFormat

7.7.1.3.23. ToBitmap Method

Creates a `Bitmap`.

```
public Bitmap ToBitmap();
```

Return Values

A `Bitmap` object.

See Also

NImage

7.7.1.3.24. ToHBitmap Method

Creates Windows HBITMAP.

```
public IntPtr ToHBitmap();
```

Return Values

A Windows HBITMAP.

See Also

NImage

7.7.1.4. NImageFormat Class

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

Fields

Bmp	Specifies the BMP image format.
Formats	Specifies collection of supported image formats.
Gif	Specifies the GIF image format.
Jpeg	Specifies the JPEG image format.
Png	Specifies the PNG image format.
Tiff	Specifies the TIFF image format.

Properties

CanRead	Gets a value indicating whether the current image format supports reading.
CanWrite	Gets a value indicating whether the current image format supports writing.
DefaultFileExtension	Gets default file extension of the current im-

	age format.
FileFilter	Gets file filter of the current image format.
Name	Gets name of the current image format.

Methods

LoadImage	Loads `NImage`.
SaveImage	Saves `NImage`.
Select	Retrieves supported image format registered with file extension of specified file name and supporting reading/writing as specified.

7.7.1.4.1. Bmp Field

```
public static readonly NImageFormat Bmp;
```

7.7.1.4.2. Formats Field

```
public static readonly NImageFormat.ImageFormatCollection Formats;
```

7.7.1.4.3. Gif Field

```
public static readonly NImageFormat Gif;
```

7.7.1.4.4. Jpeg Field

```
public static readonly NImageFormat Jpeg;
```

7.7.1.4.5. Png Field

```
public static readonly NImageFormat Png;
```

7.7.1.4.6. Tiff Field

```
public static readonly NImageFormat Tiff;
```

7.7.1.4.7. CanRead Property

Gets a value indicating whether the current image format supports reading.

```
public virtual bool CanRead {get;}
```

Property value

`true` if image format can read, `false` if image format can not read.

See Also

CanWrite | Name | DefaultFileExtension | FileFilter

7.7.1.4.8. CanWrite Property

Gets a value indicating whether the current image format supports writing.

```
public virtual bool CanWrite {get;}
```

Property value

`true` if image format can write, `false` if image format can not write.

See Also

CanRead | Name | DefaultFileExtension | FileFilter

7.7.1.4.9. DefaultFileExtension Property

Gets default file extension of the current image format.

```
public virtual string DefaultFileExtension {get;}
```

Property value

Default file extension.

See Also

CanRead | CanWrite | Name | DefaultFileExtension | FileFilter

7.7.1.4.10. FileFilter Property

Gets file filter of the current image format.

```
public virtual string FileFilter {get;}
```

Property value

An image format file filter

See Also

CanRead | CanWrite | Name | DefaultFileExtension

7.7.1.4.11. Name Property

Gets name of the current image format.

```
public virtual string Name {get;}
```

Property value

An image format name.

See Also

CanRead | CanWrite | FileFilter | DefaultFileExtension

7.7.1.4.12. LoadImage Method

Loads `NImage`.

7.7.1.4.12.1. LoadImage(string)

Loads `NImage` from file.

```
public virtual NImage LoadImage(
        string fileName
);
```

Parameters

`fileName`	A string that contains the name of the file.

Return Values

A NImage object.

See Also

NImage | SaveImage

7.7.1.4.12.2. LoadImage(IntPtr, int)

Loads `NImage` from memory buffer.

```
public virtual NImage LoadImage(
        IntPtr buffer,
        int bufferLength
);
```

Parameters

`buffer`	Pointer to memory buffer.
`bufferLength`	Size of memory buffer.

Return Values

A NImage object.

See Also

NImage | SaveImage

7.7.1.4.12.3. LoadImage(byte[])

Loads `NImage` from byte array.

```
public virtual NImage LoadImage(
        byte[] buffer
);
```

Parameters

`buffer`	A byte array.

Return Values

A NImage object.

See Also

NImage | SaveImage

7.7.1.4.13. SaveImage Method

Saves `NImage`.

7.7.1.4.13.1. void SaveImage (NImage, string)

Saves NImage to file.

```
public virtual void SaveImage(
        NImage image,
        string fileName
);
```

Parameters

image	A NImage object.
fileName	A string that contains the name of the file.

See Also

LoadImage | NImage

7.7.1.4.13.2. byte[] SaveImage(NImage)

Saves NImage to byte array.

```
public virtual byte[] SaveImage(
        NImage image
);
```

Parameters

image	A NImage object.

Return Values

A byte array.

See Also

LoadImage | NImage

7.7.1.4.13.3. void SaveImage(NImage, Stream)

Saves NImage to stream.

```
public virtual void SaveImage(
        NImage image,
        Stream stream
);
```

Parameters

`image`	A NImage object.
`stream`	The data stream used to save the image.

See Also

LoadImage | NImage

7.7.1.4.14. Select Method

Retrieves supported image format registered with file extension of specified file name and supporting reading/writing as specified.

```
public static NImageFormat Select(
        string fileName,
        FileAccess fileAccess
);
```

Parameters

`fileName`	A string that contains the name of the file.
`fileAccess`	A file access.

Return Values

A NImageFormat object.

See Also

NImageFormat

7.7.1.5. NImageFormat.ImageFormatCollection Class

Represents the collection of formats in a NImageFormat.

Properties

Item	Gets the member from collection by index.

Methods

IndexOf	Returns the index within the collection of the specified image format.

7.7.1.5.1. ImageFormatCollection.Item Property

Gets the member from collection by index.

```
public NImageFormat this[
        int index
] {get;}
```

Parameters

x	The index of the element to get.

Property value

A NImageFormat object.

See Also

NImageFormat

7.7.1.5.2. IndexOf Method

Returns the index within the collection of the specified image format.

```
public int IndexOf(
        NImageFormat value
);
```

Parameters

value	A NImageFormat object.

Return Values

The zero-based index of the NImageFormat in the collection.

See Also

NImageFormat

7.7.1.6. NImages Class

Provides library registration and other additional functionality.

Properties

Methods

GetGrayscaleColorWrapper	Creates NImage object wrapper.

Constants

DllName	Name of DLL containing unmanaged part of this class.

7.7.1.6.1. IsRegistered Property

Checks if Neurotec.NImages Pro library is registered.

```
public static bool IsRegistered {get;}
```

Property value

`true` if library is registered, `false` if library is not registered.

7.7.1.6.2. GetGrayscaleColorWrapper Method

Creates NImage object wrapper.

```
public static NImage GetGrayscaleColorWrapper(
        NImage image,
        NRgb minColor,
        NRgb maxColor
);
```

Parameters

`image`	A NImage object.
`minColor`	Specifies color to be used for black color.
`maxColor`	Specifies color to be used for white color.

Return Values

An NImage object.

Remarks

Gray values in source image are replaced with according RGB values from range [*`minCol-`*

`or`, *`maxColor`*] in created image.

See Also

NImage

7.7.1.7. NMonochromeImage Class

Provides functionality for managing 1-bit monochrome images.

Properties

Item	Gets or sets the color of the specified pixel in NImage object.

Remarks

This class provides advanced functionality, such as individual pixel value retrieval for image with pixel format equal to Monochrome.

7.7.1.7.1. NMonochromeImage.Item Property

Gets or sets the color of the specified pixel in NImage object.

```
public bool this[
        uint x,
        uint y
] {get; set;}
```

Parameters

x	The x coordinate of the pixel.
y	The y coordinate of the pixel.

Property value

If pixel is black then gets/sets `false` and if it is white then gets/sets `true`.

See Also

NImage

7.7.1.8. NPixelFormat Struct

Provides functionality for work with pixel format.

Fields

Grayscale	Each pixel value is stored in 8 bits representing 256 shades of gray.
Monochrome	Each pixel value is stored in 1 bit representing either black or white color.
Rgb	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 these fields. However only these fields are provided for usage with this SDK.

Properties

BitsPerPixel	Gets number of bits used to store a pixel from NPixelFormat Fields.

Methods

CalcRowLongSize	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.
CalcRowSize	Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.
Equals	Determines whether the specified Object is equal to the current Object.
GetHashCode	Is intended for a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetRowLongSize	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.
GetRowSize	Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.

IsValid	Checks whether current NPixelFormat value is valid.

7.7.1.8.1. Grayscale Field

Each pixel value is stored in 8 bits representing 256 shades of gray.

```
public static readonly NPixelFormat Grayscale;
```

See Also

NPixelFormat

7.7.1.8.2. Monochrome Field

Each pixel value is stored in 1 bit representing either black or white color.

```
public static readonly NPixelFormat Monochrome;
```

See Also

NPixelFormat

7.7.1.8.3. Rgb Field

Each pixel value is stored in 24 bits consisting of three 8-bit values representing red, green and blue color components.

```
public static readonly NPixelFormat Rgb;
```

See Also

NPixelFormat

7.7.1.8.4. BitsPerPixel Property

Gets number of bits used to store a pixel from NPixelFormat Fields.

```
public uint BitsPerPixel {get;}
```

Property value

A number of bits.

See Also

NPixelFormat Fields

7.7.1.8.5. CalcRowLongSize Methods

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.

7.7.1.8.5.1. CalcRowLongSize (uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.

```
public static ulong CalcRowLongSize(
        uint bitCount,
        uint length
);
```

Return Values

The number of bytes needed to store line of specified length of pixels with specified bits per pixel.

See Also

CalcRowLongSize

7.7.1.8.5.2. CalcRowLongSize (uint, uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel and alignment.

```
public static ulong CalcRowLongSize(
        uint bitCount,
        uint length,
        uint alignment
);
```

Return Values

The number of bytes needed to store line of specified length of pixels with specified bits per pixel.

See Also

CalcRowLongSize

7.7.1.8.6. CalcRowSize Methods

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.

7.7.1.8.6.1. CalcRowSize (uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel.

```
public static uint CalcRowSize(
        uint bitCount,
        uint length
);
```

Return Values

The number of bytes needed to store line of specified length of pixels with specified bits per pixel.

See Also

CalcRowSize

7.7.1.8.6.2. CalcRowSize (uint, uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified bits per pixel and alignment.

```
public static uint CalcRowSize(
        uint bitCount,
        uint length,
        uint alignment
);
```

Return Values

The number of bytes needed to store line of specified length of pixels with specified bits per pixel.

See Also

CalcRowSize

7.7.1.8.7. Equals Method

Determines whether the specified Object is equal to the current Object.

```
public override bool Equals(
        object obj
);
```

Parameters

`obj`	The Object to compare with the current Object.

Return Values

`true` if the specified Object is equal to the current Object; otherwise, `false`.

See Also

NPixelFormat

7.7.1.8.8. GetHashCode Method

Is intended for a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.

```
public override int GetHashCode();
```

Return Values

A hash code for the current Object.

7.7.1.8.9. GetRowLongSize Method

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.

7.7.1.8.9.1. GetRowLongSize (uint)

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.

```
public ulong GetRowLongSize(
        uint length
);
```

Return Values

The number of bytes needed to store line of specified length of pixels with specified NPixelFormat.

See Also

GetRowLongSize

7.7.1.8.9.2. GetRowLongSize (uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat and alignment.

```
public ulong GetRowLongSize(
        uint length,
        uint alignment
);
```

Return Values

The number of bytes needed to store line of specified length of pixels with specified NPixelFormat.

See Also

GetRowLongSize

7.7.1.8.10. GetRowSize Method

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.

7.7.1.8.10.1. GetRowSize (uint)

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat.

```
public uint GetRowSize(
        uint length
);
```

Return Values

The number of bytes needed to store line of specified length of pixels with specified NPixelFormat.

See Also

GetRowSize

7.7.1.8.10.2. GetRowSize (uint, uint)

Calculates number of bytes needed to store line of specified length of pixels with specified NPixelFormat and alignment.

```
public uint GetRowSize(
        uint length,
        uint alignment
);
```

Return Values

The number of bytes needed to store line of specified length of pixels with specified NPixelFormat.

See Also

GetRowSize

7.7.1.8.11. IsValid Method

Checks whether current NPixelFormat value is valid.

```
public static bool IsValid(
        NPixelFormat value
);
```

Parameters

`value`	The NPixelFormat object.

Return Values

Returns `true` if the object is valid NPixelFormat, `false` if not.

7.7.1.9. NRgb Struct

Represents an RGB color.

Constructors

NRgb	Initializes a new instance of the NRgb structure.

Properties

Blue	Gets the blue component value of this NRGB structure.
Green	Gets the green component value of this NRGB structure.
Red	Gets the red component value of this NRGB

	structure.

7.7.1.9.1. NTgb constructor

Initializes a new instance of the NRgb structure.

```
public NRgb(
        byte red,
        byte green,
        byte blue
);
```

Parameters

red	The blue component value of this NRGB.
green	The green component value of this NRGB.
blue	The red component value of this NRGB.

7.7.1.9.2. Blue Property

Gets the blue component value of this NRGB structure.

```
public byte Blue {get; set;}
```

Property value

The blue component value of this NRGB.

7.7.1.9.3. Green Property

Gets the green component value of this NRGB structure.

```
public byte Green {get; set;}
```

Property value

The green component value of this NRGB.

7.7.1.9.4. Red Property

Gets the red component value of this NRGB structure.

```
public byte Red {get; set;}
```

Property value

The red component value of this NRGB.

7.7.1.10. NRgbImage Class

Provides functionality for managing 24-bit RGB images.

Properties

Item	Gets the pixel by index.

Remarks

This class provides advanced functionality, such as individual pixel value retrieval for image with pixel format equal to Rgb.

7.7.1.10.1. NRgbImage.Item Property

Gets the pixel by index.

```
public NRgb this[
        uint x,
        uint y
] {get; set;}
```

Property value

A NRgb structure.

Parameters

x	The x coordinate to get or set.
y	The y coordinate to get or set.

See Also

NRgb

7.7.1.11. Tiff Class

Provides functionality for loading and saving images in TIFF format.

Constructors

Tiff	Initializes a new instance of the `Tiff` class.

Methods

LoadImage	Creates NImage object.

7.7.1.11.1. Tiff Constructor

Initializes a new instance of the `Tiff`.

```
public Tiff();
```

See Also

`Tiff` Class

7.7.1.11.2. LoadImage Method

Creates NImage object.

7.7.1.11.2.1. LoadImage (string)

Creates NImage object from TIFF file.

```
public static NImage LoadImage(
        string fileName
);
```

Parameters

`fileName`	A string that contains the name of the file.

Return Values

A NImage object.

See Also

NImage | LoadImage

7.7.1.11.2.2. LoadImage (IntPtr, int)

Creates NImage object from memory buffer.

```
public static NImage LoadImage(
        IntPtr buffer,
        int bufferLength
);
```

Parameters

buffer	Pointer to memory buffer.
bufferLength	Size of memory buffer.

Return Values

A NImage object.

See Also

NImage | LoadImage

7.7.1.11.2.3. LoadImage (byte[])

Creates NImage object from byte array.

```
public static NImage LoadImage(
        byte[] buffer
);
```

Parameters

buffer	A byte array.

Return Values

A NImage object.

See Also

NImage | LoadImage

Chapter 8. Obsolete

The Section 8.2, “VeriFinger library” and Section 8.3, “ScanMan library” libraries functions are **obsolete**. Use new libraries Chapter 6, *Reference (C/C++)* and Chapter 7, *Reference (.NET)* in .NET instead of them. See also samples located `samles/dotNET/`.

8.1. Fingerprint images

Fingerprint image used by VeriFinger library and ScanMan 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.

In Visual Basic (6.0, .Net) and MS Access images are stored in arrays with lower bounder `0` and upper bounder `width*height-1`, all elements are bytes with value from 0 to 255 (from black to white) and represent one pixel. Lines of the image have to be stored in the array from top to bottom order. For all functions that required image as parameter this image array must be passed.

8.2. VeriFinger library

VeriFinger library is a fingerprint recognition engine that you can use in your application to implement user enrollment, verification and identification using fingerprint images. It provides a number of functions covering different usage scenarios.

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 features 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.

VeriFinger library is copy protected. To use it you have to register it. See Registration.

Before using the library it has to be initialized. See Initialization and Contexts.

VeriFinger library behavior is controlled through parameters.

8.2.1. Library functions

VeriFinger library contains the following functions grouped by categories:

Registration	
VFRegistrationType	Returns registration type of VeriFinger library
VFGenerateId	Generates registration id from serial number
VFRegister	Registers VeriFinger library
Initialization	
VFInitialize	Initializes VeriFinger library
VFFinalize	Uninitializes VeriFinger 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 indicates error code.

VeriFinger Java, C# wrappers functions do not return results code but throw exception (`VeriFingerException`) when error occurs. You can get error code using `getErrorCode` method for Java and property ErrorCode for C#.

Each function (except for registration, initialization and features functions) takes last argument of type `HVFCONTEXT`. It is the context in which VeriFinger library functions are called. Pass `null` (`VF_DEFAULT_CONTEXT` or `0`, in Java and Visual Basic) to use default context. For more information see Initialization and Contexts.

You can use `VFFailed` and `VFSucceeded` functions to determine if the execution of the function failed or succeeded:

C:

```
#define VFFailed(result) ...
#define VFSucceeded(result) ...
```

Delphi:

```
function VFSucceeded(Res: Integer): Boolean;
function VFFailed(Res: Integer): Boolean;
```

Visual Basic:

```
Public Function VFSucceeded(ByVal result As Long) As Boolean
Public Function VFFailed(ByVal result As Long) As Boolean
```

Visual Basic .Net:

```
Public Function VFSucceeded(ByVal result As Integer) As Boolean
Public Function VFFailed(ByVal result As Integer) As Boolean
```

Java:

```
public class VeriFingerException extends Exception {
...
public static boolean failed(int errorCode) ...
public static boolean succeeded(int errorCode) ...
...
}
```

8.2.2. 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	VeriFinger 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
Registration		
`VFE_NOT_REGISTERED`	-2000	VeriFinger library is not registered

`VFE_INVALID_SERIAL_NUMBER`	-2001	Specified serial number is invalid
`VFE_INVALID_REGISTRATION_KEY`	-2002	Specified registration key is invalid
`VFE_SCANNER_DRIVER_ERROR`	-2003	Scanner driver error
`VFE_REGISTRATION_NOT_NEEDED`	-2004	No need to register VeriFinger library
`VFE_NO_SCANNER`	-2005	No scanner found
`VFE_MORE_THAN_ONE_SCANNER`	-2006	More than one scanner found
`VFE_LM_CONNECTION_ERROR`	-2007	Error communicating with License Manager
`VFE_LM_NO_MORE_LICENCES`	-2008	No more License Manager licenses are available
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 quality is low
VeriFinger 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 VeriFinger library. For C they are implemented in `VFinger.h` and `VFingerX.cpp` files, Delphi - in `VFinger.pas` module, Visual Basic 6.0 - in `VFinger.bas`, Visual Basic .Net - in `VFinger.vb`, Access - in VFinger module, Java - in VeriFingerException class, C# - in VeriFingerExcepton class. To get error description in Java you also can use VeriFingerException class `getMessage` method, in c# - property Message.

C:

```
string VFErrorToString(INT error);
string VFResultToString(INT result);
void VFRaiseError(INT error);
void VFCheckResult(INT result);
```

Delphi:

```
function VFErrorToString(Err: Integer): string;
function VFResultToString(Res: Integer): string;
procedure VFRaiseError(Err: Integer);
procedure VFCheckResult(Res: Integer);
```

Visual Basic:

```
Public Function VFErrorToString(ByVal code As Long) As String
Public Function VFResultToString(ByVal result As Long) As String
Public Sub VFCheckResult(ByVal result As Long)
```

Visual Basic .Net:

```
Public Function VFErrorToString(ByVal code As Integer) As String
Public Function VFResultToString(ByVal result As Integer) As String
Public Sub VFCheckResult(ByVal result As Integer)
```

Java:

```
public class VeriFingerException extends Exception {
...
public static String VFErrorToString(int error) ...
public static String VFResultToString(int result) ...
...
```

C#:

```
public class VeriFingerException: Exception{
...
public int ErrorCode
{
        get;
}
public override string Message
{
        get;
}
```

8.2.3. Registration

You have to register VeriFinger 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, registration to U.are.U scanner 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 VeriFinger library.

If you are using registration to PC or U.are.U scanner then call VFGenerateId function (for registration with U.are.U scanner connect the scanner before calling the function) and pass serial number provided with your VeriFinger 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 VeriFinger library is registered (and if it needs registration at all) call VFRegistrationType function.

Example:

C:

```
// Registration to PC or to U.are.U scanner
// 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 VeriFinger library
VFRegister(serial_number, registration_key);
```

Delphi:

```
// Registration to PC or to U.are.U scanner
// Your serial number here
const SerialNumber = 'xxxx-xxxx-xxxx-xxxx';
// Registration id generation
var
        RegistrationId: string;
begin
        SetLength(RegistrationId, 100);
        VFGenerateId(SerialNumber, RegistrationId);
end;
// Received registration key
const
        RegistrationKey = 'xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx';
        // Register VeriFinger library
begin
        VFRegister(SerialNumber, RegistrationKey);
end;
```

Visual Basic 6.0/.Net:

```
' Registration to PC or to U.are.U scanner
' Your serial number here
Dim SerialNumber As String
SerialNumber = "xxxx-xxxx-xxxx-xxxx"
' Registration id generation
Dim RegistrationId As String
' Error code
Dim ErrCode As Long ' use "Integer" in Visual Basic .Net
RegistrationId = Space(100)
ErrCode = VFGenerateId(SerialNumber, RegistrationId)
' Received registration key
Dim RegistrationKey As String
RegistrationKey = "xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx"
' Register VeriFinger library
ErrCode = VFRegister(SerialNumber, RegistrationKey)
```

Java:

```
// Registration to PC or to U.are.U scanner
// Your serial number here
String SerialNumber = "xxxx-xxxx-xxxx-xxxx";
// Registration id generation
String RegistrationId = "";
try {
        RegistrationId = VeriFingerWrapper.VFGenerateId(SerialNumber);
} catch (VeriFingerException vfe) {
// error handler code
} catch (Exception ex) {
}
// Received registration key
String RegistrationKey = "xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx";
// Register VeriFinger library
try {
        VeriFingerWrapper.VFRegister(SerialNumber, RegistrationKey);
} catch (VeriFingerException vfe) {
// error handler code
} catch (Exception ex) {
}
```

C#:

```
/ Registration to PC or to U.are.U scanner
// Your serial number here
string serialNumber = "xxxx-xxxx-xxxx-xxxx";
// Registration id generation
string registrationId = "";
try
{
        registrationId = veriFinger.GenerateId(serialNumber);
} catch(VeriFingerException ex)
{}
// Received registration key
string registrationKey = "xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx";
// Register VeriFinger library
try
{
        veriFinger.Register(serialNumber, registrationKey);
} catch(VeriFingerException ex)
{}
```

8.2.3.1. VFRegistrationType function

Returns VeriFinger library registration type.

C:

```
INT VFINGER_API VFRegistrationType();
```

Delphi:

```
function VFRegistrationType: Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFRegistrationType
        Lib "VFVBP42.dll"
        Alias "VBVFRegistrationType" () As Long
```

Visual Basic .Net:

```
Public Declare Function VFRegistrationType
        Lib "VFVBP42.dll" Alias
        "VBVFRegistrationType" () As Integer
```

Java:

```
public static native int VFRegistrationType()
throws VeriFingerException, Exception;
```

C#:

```
public int RegistrationType{get;} throws VeriFingerException;
```

Return values:

`VF_RT_NOT_PROTECTED`	0	VeriFinger 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	VeriFinger library is registered to PC
`VF_RT_UAREU`	4	VeriFinger library is registered to U.are.U scanner
`VF_RT_LAN`	8	VeriFinger library is registered in License Manager on LAN
`VF_RT_UNREGISTERED`	6	VeriFinger library is not registered. Call VFRegister function to register

In case of error functions returns `VFE_FAILED`, VeriFinger Java, C# wrappers throw exceptions.

8.2.3.2. 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);
```

Delphi:

```
function VFGenerateId(Serial, Id: PChar): Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFGenerateId
        Lib "VFVBP42.dll"
        Alias "VBVFGenerateId"
        (
                ByVal Serial As String,
                ByVal id As String
        ) As Long
```

Visual Basic .Net:

```
Public Declare Function VFGenerateId
        Lib "VFVBP42.dll"
        Alias "VBVFGenerateId"
        (
                ByVal Serial As String,
                ByVal id As String
        ) As Integer
```

Java:

```
// Returns registration id
public static native String VFGenerateId(String serial)
throws VeriFingerException, Exception;
```

C#:

```
// Returns registration id
public string GenerateId(string serial) throws VeriFingerException;
```

Parameters:

[in]	serial, Serial	Serial number of VeriFinger 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`.If serial number indicates registration to DigitalPersona U.are.U scanner then can return `VFE_SCANNER_DRIVER_ERROR` (if there was error communicating with scanner driver), `VFE_NO_SCANNER` (if no scanner detected) or `VFE_MORE_THAN_ONE_SCANNER` (if more than one scanner detected).Otherwise generates registration id and returns `VFE_OK` (in case of error returns `VFE_FAILED`). VeriFinger Java, C# wrappers throw exceptions if error occurs.

8.2.3.3. VFRegister function

Registers VeriFinger 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);
```

Delphi:

```
function VFRegister(Serial, Key: PChar): Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFRegister
        Lib "VFVBP42.dll"
        Alias "VBVFRegister"
        (
                ByVal Serial As String,
                ByVal Key As String
        ) As Long
```

Visual Basic .Net:

```
Public Declare Function VFRegister
        Lib "VFVBP42.dll"
        Alias "VBVFRegister"
        (
                ByVal Serial As String,
                ByVal Key As String
        ) As Integer
```

Java:

```
public static native void VFRegister(String serial, String key)
throws VeriFingerException, Exception;
```

C#:

```
public int Register(string serial, string key) throws VeriFingerException;
```

Parameters:

[in]	serial, Serial	Serial number of VeriFinger library license. If you have VeriFinger 4.0 PC protection registration key please use customer id instead of serial number. For example: customer id = 11222 then serial will be `"11222"` 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 have VeriFinger 4.0 PC protection registration key please convert it to string and pass as key. If you are using LAN protection then you must specify server name as registration key.

Return values: If VeriFinger 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`.If registration key is invalid (or scanner is not connected for registration to U.are.U scanner) returns `VFE_INVALID_REGISTRATION_KEY`.Otherwise registers VeriFinger library and returns `VFE_OK` (in case of error returns `VFE_FAILED`). VeriFinger Java, C# wrappers throw exceptions if error occurs.

8.2.4. Initialization

VeriFinger 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 your custom context.

See Contexts for more information.

Example:

C:

```
// Main application function
{
        // Application initialization code
        VFInitialize();
        // Other application code
        VFFinalize();
        // Application uninitialization code
}
```

Delphi:

```
// In project source
begin
        // Application initialization code
        VFInitialize;
        // Other application code
        VFFinalize;
        // Application uninitialization code
end.
```

Visual Basic 6.0/.Net:

```
' In project source which is using Main sub as startup object
Sub Main()
        ' Application initialization code
        VFInitialize
        ' Other application code
        VFFinalize
        ' Application uninitialization code
        End Sub
        ' In project source which is using form as startup object
        Private Sub Form_Load...
        VFInitialize
        ' Application initialization code
        End Sub
        ' Other application code
        Private Sub Form_Unload...
        ' Application uninitialization code
        VFFinalize
End Sub
```

Java:

```
// Main
{
        // Application initialization code
        try {
                VeriFingerWrapper.VFInitialize();
        } catch (VeriFingerException vfe) {
        // error handling code
        } catch (Exception e) {
        // error handling code
        }
        // Other application code
        try {
                VeriFingerWrapper.VFFinalize();
        } catch (VeriFingerException vfe) {
        // error handling code
        } catch (Exception e) {
        // error handling code
        }
        // Application uninitialization code
}
```

C#:

```
try
{
        VeriFinger veriFinger = new VeriFinger();
}catch(VeriFingerException ex)
{}

try
{
        veriFinger.Dispose();
}catch(VeriFingerException ex)
{}
```

8.2.4.1. VFInitialize function

Creates default context by calling VFCreateContext function and initializes VeriFinger library.

C:

```
INT VFINGER_API VFInitialize();
```

Delphi:

```
function VFInitialize: Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFInitialize Lib "VFVBP42.dll"
Alias "VBVFInitialize" () As Long
```

Visual Basic .Net:

```
Public Declare Function VFInitialize Lib "VFVBP42.dll"
Alias "VBVFInitialize" () As Integer
```

Java:

```
public static native int VFInitialize()
throws VeriFingerException, Exception;
```

C#:

```
try
{
        VeriFinger veriFinger = new VeriFinger();
}catch(VeriFingerException ex)
{}
```

Return values: If succeeded return value indicates number of times function have been called before. If it first call to the function return value will be zero.If default context was not created returns `VFE_OUT_OF_MEMORY`. VeriFinger Java wrapper throws exception if error occurs.

8.2.4.2. VFFinalize function

Destroys default context by calling VFFreeContext function and uninitializes VeriFinger library if call to the function corresponds to first call to VFInitialize function.

C:

```
INT VFINGER_API VFFinalize();
```

Delphi:

```
function VFFinalize: Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFFinalize Lib "VFVBP42.dll"
        Alias "VBVFFinalize" () As Long
```

Visual Basic .Net:

```
Public Declare Function VFFinalize Lib "VFVBP42.dll"
        Alias "VBVFFinalize" () As Integer
```

Java:

```
public static native int VFFinalize() throws VeriFingerException, Exception;
```

C#:

```
try
{
        veriFinger.Dispose();
}catch(VeriFingerException ex)
{}
```

Return values: Return value indicates number of times function should be more called (number of `VFInitialize` calls without `VFFinalize` calls).If VeriFinger library was not initialized returns `VFE_NOT_INITIALIZED`. VeriFinger Java wrapper throws exception if error occurs.

8.2.5. Contexts

Context is a set of parameters and internal structures that VeriFinger library functions use. They are created with VFCreateContext function and destroyed with VFFreeContext function.

Contexts enable different application parts to work with VeriFinger library simultaneously. Inside one context no VeriFinger functions should be called simultaneously because they are not guaranteed to be thread-safe. VeriFinger 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 VeriFinger 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`.

Example: Working from different threads:

C:

```
// First thread function
{
        // Create context
        HVFCONTEXT context = VFCreateContext();
        // Call VeriFinger library functions, for example
        VFVerify(..., context);
        // Delete context
        VFFreeContext(context);
}
// Second thread function
{
        // Create context
        HVFCONTEXT context = VFCreateContext();
        // Call VeriFinger library functions, for example
        VFIdentifyNext(..., context);
        // Delete context
        VFFreeContext(context);
}
```

Delphi:

```
// First thread method
var
        Context: HVFCONTEXT;
begin
        // Create context
        Context := VFCreateContext;
        // Call VeriFinger library functions, for example
        VFVerify(..., Context);
        // Delete context
        VFFreeContext(Context);
end;
// Second thread method
var
        Context: HVFCONTEXT;
begin
        Context := VFCreateContext;
        // Call VeriFinger library functions, for example
        VFIdentifyNext(..., Context);
        // Delete context
        VFFreeContext(Context);
end;
```

Visual Basic:

```
' First thread code
Dim ErrCode as Long
Dim Context as Long
' Create context
```

```
Context = VFCreateContext()
' Call VeriFinger library functions, for example
ErrCode = VFVerify(..., Context)
' Delete context
ErrCode = VFFreeContext(Context)
' Second thread code
Dim ErrCode as Long
Dim Context as Long
Context = VFCreateContext()
' Call VeriFinger library functions, for example
ErrCode = VFIdentifyNext(..., Context)
' Delete context
ErrCode = VFFreeContext(Context)
```

Visual Basic .Net:

```
' First thread code
Dim ErrCode as Integer
Dim Context as Integer
' Create context
Context = VFCreateContext()
' Call VeriFinger library functions, for example
ErrCode = VFVerify(..., Context)
' Delete context
ErrCode = VFFreeContext(Context)
' Second thread code
Dim ErrCode as Integer
Dim Context as Integer
Context = VFCreateContext()
' Call VeriFinger library functions, for example
ErrCode = VFIdentifyNext(..., Context)
' Delete context
ErrCode = VFFreeContext(Context)
```

Java:

```
// First thread code fragment (exception handling code omitted)
// Create context
int context = VeriFingerWrapper.VFCreateContext();
// Call VeriFinger library functions, for example
VeriFingerWrapper.VFVerify(..., context);
// Delete context
VeriFingerWrapper.VFFreeContext(context);
// Second thread code fragment (exception handling code omitted)
// Create context
int context = VeriFingerWrapper.VFCreateContext();
// Call VeriFinger library functions, for example
VeriFingerWrapper.VFIdentifyNext(..., context);
// Delete context
VeriFingerWrapper.VFFreeContext(context);
```

C#:

```
// First thread code fragment (exception handling code omitted)
VeriFinger veriFinger = new VeriFinger();
// Call VeriFinger library functions, for example
veriFinger.Verify(features1, features2);
veriFinger.Dispose();

// Second thread code fragment (exception handling code omitted)
VeriFinger veriFinger2 = new VeriFinger();
veriFinger2.IdenfifyNext(...);
veriFinger2.Dispose();
```

Example: 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 VeriFinger library functions, for example
        VFExtract(..., context);
}
// Uninitialization function
{
        // Delete first context
        VFFreeContext(context1);
        // Delete second context
        VFFreeContext(context2);
}
```

Delphi:

```
var
```

```
        Context1: HVFCONTEXT; // First context
        Context2: HVFCONTEXT; // Second context
// Initialization function
begin
        // Set parameters for default context
        VFSetParameter(..., nil);
        VFSetParameter(..., nil);
        // 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(..., Context);
        VFSetParameter(..., Context);
end;
// Some application function
var
        Context: HVFCONTEXT;
begin
        if { image from first scanner } then Context := Context1
        else
                if { image from second scanner } then Context := Context2
                else Context := nil; // default context
        // Call VeriFinger library functions, for example
        VFExtract(..., Context);
end;
// Uninitialization function
begin
        // Delete first context
        VFFreeContext(Context1);
        // Delete second context
        VFFreeContext(Context2);
end;
```

Visual Basic:

```
Dim context1 as Long ' First context
Dim context2 as Long ' Second context
Dim ErrCode as Long
' Initialization function
' Set parameters for default context
ErrCode = VFSetParameter(..., VF_DEFAULT_CONTEXT)
ErrCode = VFSetParameter(..., VF_DEFAULT_CONTEXT)
' Create first context
context1 = VFCreateContext()
' Set parameters for first context
ErrCode = VFSetParameter(..., context1)
ErrCode = VFSetParameter(..., context1)
' Create second context
context2 = VFCreateContext()
' Set parameters for second context
ErrCode = VFSetParameter(..., context2)
ErrCode = VFSetParameter(..., context2)
' Some application function
Dim context as long
if ' image from first scanner
        context = context1;
else
        if ' image from second scanner
                context = context2
        else
                context = VF_DEFAULT_CONTEXT ' default context
        end if
end if
' Call VeriFinger library functions, for example
ErrCode = VFExtract(..., context)
' Uninitialization function
' Delete first context
ErrCode = VFFreeContext(context1)
' Delete second context
ErrCode = VFFreeContext(context2)
```

Visual Basic .Net:

```
Dim context1 as Integer ' First context
Dim context2 as Integer ' Second context
Dim ErrCode as Integer
' Initialization function
' Set parameters for default context
ErrCode = VFSetParameter(..., VF_DEFAULT_CONTEXT)
ErrCode = VFSetParameter(..., VF_DEFAULT_CONTEXT)
' Create first context
context1 = VFCreateContext()
' Set parameters for first context
ErrCode = VFSetParameter(..., context1)
ErrCode = VFSetParameter(..., context1)
' Create second context
context2 = VFCreateContext()
' Set parameters for second context
ErrCode = VFSetParameter(..., context2)
ErrCode = VFSetParameter(..., context2)
' Some application function
Dim context as Integer
if ' image from first scanner
        context = context1;
else
        if ' image from second scanner
                context = context2
        else
                context = VF_DEFAULT_CONTEXT ' default context
        end if
end if
' Call VeriFinger library functions, for example
ErrCode = VFExtract(..., context)
' Uninitialization function
' Delete first context
ErrCode = VFFreeContext(context1)
```

```
' Delete second context
ErrCode = VFFreeContext(context2)
```

Java:

```
int context1; // First context
int context2; // Second context
// Initialization function (exception handling code omitted)
{
        // Set parameters for default context
        VeriFingerWrapper.VFSetParameter(..., VF_DEFAULT_CONTEXT);
        VeriFingerWrapper.VFSetParameter(..., VF_DEFAULT_CONTEXT);
        // Create first context
        context1 = VeriFingerWrapper.VFCreateContext();
        // Set parameters for first context
        VeriFingerWrapper.VFSetParameter(..., context1);
        VeriFingerWrapper.VFSetParameter(..., context1);
        // Create second context
        context2 = VeriFingerWrapper.VFCreateContext();
        // Set parameters for second context
        VeriFingerWrapper.VFSetParameter(..., context2);
        VeriFingerWrapper.VFSetParameter(..., context2);
}
// Some application function (exception handling code omitted)
{
HVFCONTEXT context;
        if (/* image from first scanner */) context = context1;
                else
                    if(/* image from second scanner */) context = context2;
                    else context = VF_DEFAULT_CONTEXT; // default context
        // Call VeriFinger library functions, for example
        VeriFingerWrapper.VFExtract(..., context);
}
// Uninitialization function
{
        // Delete first context
        VeriFingerWrapper.VFFreeContext(context1);
        // Delete second context
        VeriFingerWrapper.VFFreeContext(context2);
}
```

C#:

```
{
        //Create objects
        VeriFinger veriFinger = new VeriFinger();
        veriFinger.SetParameter(...);
        veriFinger.SetParameter(...);
        VeriFinger veriFinger2 = new VeriFinger();
        veriFinger2.SetParameter(...);
        veriFinger2.SetParameter(...);
}
// Some application function (exception handling code omitted)
```

```
{
        if (/* image from first scanner */)
        {
                //use object veriFinger
        }
        else
        {
                if(/* image from second scanner */)
                {
                        //use object veriFinger2
                }
                ...
        }
}
//dispose resources(free context)
{
        veriFinger.Dispose();
        veriFinger2.Dispose();
}
```

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
}
```

Delphi:

```
// Some application function
begin
        //...
        VFIdentifyStart(...);
        for (...)
        VFIdentifyNext(...);
        VFVerify(...); // Error, returns VFE_INVALID_MODE
        VFIdentifyEnd(...);
        //...
        VFExtract(...);
        VFIdentifyNext(...); // Error, returns VFE_INVALID_MODE
end;
```

Visual Basic 6.0/.Net:

```
' Some application function/sub
' ...
VFIdentifyStart ...
For ...
VFIdentifyNext ...
Next ...
VFVerify ... // Error, returns VFE_INVALID_MODE
VFIdentifyEnd ...
' ...
VFExtract ...
VFIdentifyNext ... ' Error, returns VFE_INVALID_MODE
```

Java:

```
// Some application function (exception handling code omitted)
{
        //...
        VeriFingerWrapper.VFIdentifyStart(...);
        for (...)
        VeriFingerWrapper.VFIdentifyNext(...);
        VeriFingerWrapper.VFVerify(...); // Error,
        //exception will be thrown (VFE_INVALID_MODE)
        VeriFingerWrapper.VFIdentifyEnd(...);
        //...
        VeriFingerWrapper.VFExtract(...);
        VeriFingerWrapper.VFIdentifyNext(...); // Error,
        // expection will be thrown (VFE_INVALID_MODE)
}
```

C#:

```
// Some application function (exception handling code omitted)
}
        //...
        veriFinger.IdentifyStart(...);
        for(...)
        veriFinger.IdentifyNext(...);
        veriFinger.Verify(...);//Error
        //VeriFingerException exception will be thrown (InvalidMode)
        veriFinger.IdentifyEnd();
        //...
        veriFigner.Extract(...);
        veriFinger.IdentifyNext(...);//Error
        //VeriFingerException exception will be thrown (InvalidMode)
}
```

8.2.5.1. VFCreateContext function

Creates context with default parameters.

C:

```
HVFCONTEXT VFINGER_API VFCreateContext();
```

Delphi:

```
function VFCreateContext: HVFCONTEXT; stdcall;
```

Visual Basic:

```
Public Declare Function VFCreateContext Lib "VFVBP42.dll"
        Alias "VBVFCreateContext" () As Long
```

Visual Basic .Net:

```
Public Declare Function VFCreateContext Lib "VFVBP42.dll"
        Alias "VBVFCreateContext" () As Integer
```

Java:

```
public static native int VFCreateContext()
throws VeriFingerException, Exception;
```

C#:

```
public VeriFinger() throws VeriFingerException;
```

Return values: Return value is newly created context.If context cannot be created returns `VFE_OUT_OF_MEMORY`. VeriFinger Java wrapper throws exception if error occurs.

8.2.5.2. VFFreeContext function

Deletes context created with VFCreateContext.

C:

```
INT VFINGER_API VFFreeContext(HVFCONTEXT context);
```

Delphi:

```
function VFFreeContext(Context: HVFCONTEXT): Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFFreeContext Lib "VFVBP42.dll"
        Alias "VBVFFreeContext"
        (
                ByVal context As Long
        ) As Long
```

Visual Basic .Net:

```
Public Declare Function VFFreeContext Lib "VFVBP42.dll"
        Alias "VBVFFreeContext"
        (
                ByVal context As Integer
        ) As Integer
```

Java:

```
public static native void VFFreeContext(int context)
throws VeriFingerException, Exception;
```

C#:

```
public void Dispose() throws VeriFingerException;
```

Parameters:

	context, Context	Context to delete

Return values: If context is null returns `VFE_ARGUMENT_NULL` else returns `VFE_OK`. VeriFinger Java wrapper throws exception if error occurs.

8.2.6. Parameters

Some VeriFinger 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	Delphi equivalent	Visual Basic 6.0/.Net equivalent

Void		`VF_TYPE_VOID0`			
Byte	1	`VF_TYPE_BYTE1`	BYTE	Byte	Byte/Byte
Signed byte	1	`VF_TYPE_SBYTE2`	SBYTE	ShortInt	Byte/Byte
Word	2	`VF_TYPE_WORD3`	WORD	Word	Integer/ Short
Short integer	2	`VF_TYPE_SHORT4`	SHORT	SmallInt	Integer/ Short
Double word	4	`VF_TYPE_DWORD5`	DWORD	LongWord	Long/Integer
Integer	4	`VF_TYPE_INT6`	INT	Integer	Long/Integer
Boolean	4	`VF_TYPE_BOOL10`	BOOL	Boolean	Long/Integer
Char	1	`VF_TYPE_CHAR20`	CHAR	AnsiChar	Byte/Byte
String	4	`VF_TYPE_STRING100`	CHAR*	PAnsiChar (AnsiString)	String/ String

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.

Through Java wrapper parameters must be passed converted to String type. Wrapper returns parameters also converted to String type.

C# wrapper parameter must be passed int type. Wrapper returns parameters converted to object.

In C and Delphi 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 VeriFinger library
`VFP_VERSION_HIGH`	11	x	Double word	Major version of VeriFinger library
`VFP_VERSION_LOW`	12	x	Double word	Minor version of VeriFinger library
`VFP_COPYRIGHT`	13	x	String	Copyright of VeriFinger 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_`	0	No image is returned - the image will be the same as		

`NONE`		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_THRESHOL`

				D 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
VeriFinger specific				
VFP_MODE	1000		Integer	Specifies mode in which VeriFinger 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_	800	Authentec AFS2		

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

8.2.6.1. VFGetParameter function

Retrieves specified parameter value for specified context.

C:

```
INT VFINGER_API VFGetParameter
(
        INT parameter,
        VOID * value,
        HVFCONTEXT context
);
```

Delphi:

```
function VFGetParameter
(
        Parameter: Integer;
        Value: Pointer;
        Context: HVFCONTEXT
): Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFGetParameter Lib "VFVBP42.dll"
        Alias "VBVFGetParameter"
        (
                ByVal Parameter As Long,
                ByRef value As Variant,
                ByVal context As Long
        ) As Long
```

Visual Basic .Net:

```
Public Declare Function VFGetParameter Lib "VFVBP42.dll"
Alias "VBVFGetParameter"
(
        ByVal Parameter As Integer,
        ByRef value As Object,
        ByVal context As Integer
) As Integer
```

Java:

```
// returns parameter value converted to String
public static native String getParameterValue
(
        int parameter,
        int context
) throws VeriFingerException, Exception;
```

C#:

```
public object GetParameter(int parameter) throws VeriFingerException;
```

Parameters:

	parameter, Parameter	Parameter identifier to retrieve
[out]	value, Value	Pointer to variable that will receive parameter value. In Visual Basic case - Variant data type variable
	Context, Context	Context to retrieve parameter from. `Null` for default context (in Visual Basic case - `VF_DEFAULT_CONTEXT`)

Return values: If context is null and VeriFinger 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`.VeriFinger Java, C# wrappers throw exceptions if error occurs.

Example:

C:

```
// Some application function
{
        CHAR *name;
        INT l;
        DWORD version;
        INT vfp_mode_type;
        INT mode;
        // Get VeriFinger 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 VeriFinger library major version
        VFGetParameter(VFP_VERSION_HIGH, &version, NULL);
        printf("Version: %u.%u", 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);
}
```

Delphi:

```
// Some application function
var
        Name: AnsiString;
        L: Integer;
        Version: LongWord;
        VFPModeType: Integer;
        Mode: Integer;
begin
        // Get VeriFinger library name
        L := VFGetParameter(VFP_NAME, nil, nil);
        SetLength(Name, L + 1);
        VFGetParameter(VFP_NAME, PAnsiString(Name), nil);
        ShowMessage(Name);
        // Get VeriFinger library major version
        VFGetParameter(VFP_VERSION_HIGH, @Version, nil);
        ShowMessage(Format('Version: %u.%u', [LoWord(Version),
                                                        HiWord(Version)]));
        // Determine parameter VFP_MODE type
        VFPModeType := VFGetParameter(VFP_TYPE, Pointer(VFP_MODE), nil);
        // returned value: VF_TYPE_INT
        // Get integer parameter VFP_MODE value
        VFGetParameter(VFP_MODE, &Mode, nil);
        ShowMessage(Format('Mode: %d', [Mode]));
end;
```

Visual Basic:

```
' Some application function/sub
Dim Name As Variant
Dim Version As Variant
Dim Mode As Variant
' Get VeriFinger library name
VFGetParameter VFP_NAME, Name, VF_DEFAULT_CONTEXT
MsgBox Name
' Get VeriFinger library major version
VFGetParameter VFP_VERSION_HIGH, Version, VF_DEFAULT_CONTEXT
MsgBox "Version: " & CStr((Version And &HFFFF0000) /
        &H10000) & "." & CStr(Version And 65535)
' Get parameter VFP_MODE value
VFGetParameter VFP_MODE, Mode, VF_DEFAULT_CONTEXT
MsgBox "Mode: " & CLng(Mode)
```

Visual Basic .Net:

```
' Some application function/sub
Dim Name As Object
Dim Version As Object
Dim Mode As Object
' Get VeriFinger library name
VFGetParameter(VFP_NAME, Name, VF_DEFAULT_CONTEXT)
MsgBox(Name)
' Get VeriFinger library major version
VFGetParameter(VFP_VERSION_HIGH, Version, VF_DEFAULT_CONTEXT)
MsgBox("Version: " & CStr((Version And &HFFFF0000) /
        &H10000) & "." & CStr(Version And 65535))
' Get parameter VFP_MODE value
VFGetParameter(VFP_MODE, Mode, VF_DEFAULT_CONTEXT)
MsgBox("Mode: " & CLng(Mode))
```

Java:

```
// Some application function (exception handling code omitted)
// Get VeriFinger library name
String Name = VeriFingerWrapper.getParameterValue(VFP_NAME,
                                                VF_DEFAULT_CONTEXT);
System.out.println("Name = " + Name);
// Get parameter VFP_MODE value
String Mode = VeriFingerWrapper.getParameterValue(VFP_MODE,
                                                VF_DEFAULT_CONTEXT);
System.out.println("Mode= " + Mode);
```

C#:

```
// Some application function (exception handling code omitted)
// Get VeriFinger library name
string name = (string)veriFinger.GetParameter(VeriFinger.name)
MessageBox.Show(name);
```

```
// Get parameter VFP_MODE value
string mode = (string)veriFinger.GetParameter(VeriFinger.mode);
MessageBox.Show(mode);
```

8.2.6.2. VFSetParameter function

Sets specified parameter value for specified context.

C:

```
INT VFINGER_API VFSetParameter
(
        INT parameter,
        DWORD value,
        HVFCONTEXT context
);
```

Delphi:

```
function VFSetParameter
(
        Parameter: Integer;
        Value: LongWord;
        Context: HVFCONTEXT
): Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFSetParameter Lib "VFVBP42.dll"
Alias "VBVFSetParameter"
(
        ByVal Parameter As Long,
        ByVal value As Variant,
        ByVal context As Long
) As Long
```

Visual Basic .Net:

```
Public Declare Function VFSetParameter Lib "VFVBP42.dll"
Alias "VBVFSetParameter"
(
        ByVal Parameter As Integer,
        ByVal value As Object,
        ByVal context As Integer
) As Integer
```

Java:

```
public static native void setParameterValue
```

```
(
        int parameter,
        String value,
        int context
) throws VeriFingerException, Exception;
```

C#:

```
public int SetParameter(int parameter, bool parValue)
public int SetParameter(int parameter, int parValue)
public int SetParameter(int parameter, string parValue)
```

Parameters:

[in]	parameter, Parameter	Parameter identifier to set
[in]	value, Value	Parameter value to set
	context, Context	Context to set parameter to. `Null` for default context (in Visual Basic case - `VF_DEFAULT_CONTEXT`)

Return values: If context is `null` and VeriFinger 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`.VeriFinger Java, C# wrappers throw exceptions if error occurs.

Example:

C:

```
// Some application function
{
        // Set VFP_MODE parameter to VF_MODE_DIGITALPERSONA_URU
        VFSetParameter(VFP_MODE, (DWORD)VF_MODE_DIGITALPERSONA_URU, NULL);
}
```

Delphi:

```
// Some application function
begin
        // Set VFP_MODE parameter to VF_MODE_DIGITALPERSONA_URU
        VFSetParameter(VFP_MODE, LongWord(VF_MODE_DIGITALPERSONA_URU), nil);
end;
```

Visual Basic:

```
' Some application function
' Set VFP_MODE parameter to VF_MODE_DIGITALPERSONA_URU
Dim ErrCode as Long
ErrCode = VFSetParameter(VFP_MODE, VF_MODE_DIGITALPERSONA_URU,
                                                        VF_DEFAULT_CONTEXT)
```

Visual Basic .Net:

```
' Some application function
' Set VFP_MODE parameter to VF_MODE_DIGITALPERSONA_URU
Dim ErrCode as Integer
ErrCode = VFSetParameter(VFP_MODE, VF_MODE_DIGITALPERSONA_URU,
                                                        VF_DEFAULT_CONTEXT)
```

Java:

```
// Some application function (exception handling code omitted)
{
        // Set VFP_MODE parameter to VF_MODE_DIGITALPERSONA_URU
        VeriFingerWrapper.setParameterValue
        (
                VFP_MODE,
                Integer.toString(VF_MODE_DIGITALPERSONA_URU),
                VF_DEFAULT_CONTEXT
        );
}
```

C#:

```
// Some application function (exception handling code omitted)
//set mode parameter to modeDigitalPersonaUAreU
veriFinger.SetParameter(VeriFinger.mode,
                                VeriFinger.modeDigitalPersonaUAreU);
```

8.2.6.3. Additional functions

The following functions are not part of VeriFinger library. For C they are implemented in `VFingerX.h` and `VFingerX.cpp` files, Delphi - in `VFinger.pas` module, Visual Basic 6.0 - in `VFinger.bas` module, Visual Basic .Net - in `VFinger.vb`, Java - in VeriFingerWrapper class.

`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`).

Functions `VFGetXxxParameter` and `VFSetXxxParameter` are not available in Visual

Basic as `VFGetParameter` and `VFSetParameter` accept Variant type and typed functions are not needed.

In Java only `getParameterType` function is available.

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
BYTE VFSetByteParameter
(
        INT parameter,
        BYTE value,
        HVFCONTEXT context = NULL
);

SBYTE VFSetSByteParameter
(
        INT parameter,
        SBYTE value,
        HVFCONTEXT context = NULL
);

WORD VFSetWordParameter
(
        INT parameter,
        WORD value,
        HVFCONTEXT context = NULL
);

SHORT VFSetShortParameter
(
        INT parameter,
        SHORT value,
        HVFCONTEXT context = NULL
);

DWORD VFSetDWordParameter
(
        INT parameter,
        DWORD value,
        HVFCONTEXT context = NULL
);

INT VFSetIntParameter
(
        INT parameter,
        INT value,
        HVFCONTEXT context = NULL
);

bool VFSetBoolParameter
(
        INT parameter,
        bool value,
        HVFCONTEXT context = NULL
);

TCHAR VFSetCharParameter
(
        INT parameter,
        TCHAR value,
        HVFCONTEXT context = NULL
);

string VFSetStringParameter
(
        INT parameter,
        const string & value,
        HVFCONTEXT context = NULL
);

// Get general
INT VFGetParameterType(INT parameter);
string VFGetName();
DWORD VFGetVersionHigh();
DWORD VFGetVersionLow();
string VFGetCopyright();
```

Delphi:

```
// Get
function VFGetByteParameter
(
        Parameter: Integer;
        Context: HVFCONTEXT = nil
): Byte;

function VFGetSByteParameter
(
        Parameter: Integer;
        Context: HVFCONTEXT = nil
): ShortInt;

function VFGetWordParameter
(
        Parameter: Integer;
        Context: HVFCONTEXT = nil
): Word;

function VFGetShortParameter
(
        Parameter: Integer;
        Context: HVFCONTEXT = nil
): SmallInt;

function VFGetDWordParameter
(
        Parameter: Integer;
        Context: HVFCONTEXT = nil
): LongWord;

function VFGetIntParameter
(
        Parameter: Integer;
        Context: HVFCONTEXT = nil
): Integer;

function VFGetBoolParameter
(
        Parameter: Integer;
        Context: HVFCONTEXT = nil
): Boolean;

function VFGetCharParameter
(
        Parameter: Integer;
        Context: HVFCONTEXT = nil
): Char;

function VFGetStringParameter
(
        Parameter: Integer;
        Context: HVFCONTEXT = nil
): string;

// Set
procedure VFSetByteParameter
(
        Parameter: Integer;
        Value: Byte;
        Context: HVFCONTEXT = nil
);

procedure VFSetSByteParameter
(
        Parameter: Integer;
        Value: ShortInt;
        Context: HVFCONTEXT = nil
);

procedure VFSetWordParameter
(
        Parameter: Integer;
        Value: Word;
        Context: HVFCONTEXT = nil
);
```

```
procedure VFSetShortParameter
(
        Parameter: Integer;
        Value: SmallInt;
        Context: HVFCONTEXT = nil
);

procedure VFSetDWordParameter
(
        Parameter: Integer;
        Value: LongWord;
        Context: HVFCONTEXT = nil
);

procedure VFSetIntParameter
(
        Parameter: Integer;
        Value: Integer;
        Context: HVFCONTEXT = nil
);

procedure VFSetBoolParameter
(
        Parameter: Integer;
        Value: Boolean;
        Context: HVFCONTEXT = nil
);

procedure VFSetCharParameter
(
        Parameter: Integer;
        Value: Char;
        Context: HVFCONTEXT = nil
);

procedure VFSetStringParameter
(
        Parameter: Integer;
        Value: string;
        Context: HVFCONTEXT = nil
);

// Get general
function VFGetParameterType(Parameter: Integer): Integer;
function VFGetName: string;
function VFGetVersionHigh: LongWord;
function VFGetVersionLow: LongWord;
function VFGetCopyright: string;
```

Visual Basic:

```
' Get general
Public Declare Function VFGetParameterType Lib "VFVBP42.dll"
        Alias "VBVFGetParameterType" (ByVal Parameter As Long) As Long
Public Function VFGetName() As String
Public Function VFGetVersionHigh() As Long
Public Function VFGetVersionLow() As Long
Public Function VFGetCopyright() As String
```

Visual Basic .Net:

```
' Get general
Public Declare Function VFGetParameterType Lib "VFVBP42.dll"
        Alias "VBVFGetParameterType"(ByVal Parameter As Integer) As Integer
Public Function VFGetName() As String
Public Function VFGetVersionHigh() As Integer
Public Function VFGetVersionLow() As Integer
Public Function VFGetCopyright() As String
```

Java:

```
public static native int getParameterType(int parameter)
        throws VeriFingerException, Exception;
```

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 VeriFinger library name
        name = VFGetStringParameter(VFP_NAME);
        // or
        name = VFGetName();
        printf(name);
        // Get VeriFinger library major version
        version = VFGetDWordParameter(VFP_VERSION_HIGH);
        // or
        version = VFGetVersionHigh();
        printf("Version: %u.%u", 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);
        // Set VFP_MODE parameter to VF_MODE_DIGITALPERSONA_URU
        VFSetIntParameter(VFP_MODE, VF_MODE_DIGITALPERSONA_URU);
}
```

Delphi:

```
// Some application function
var
        Name: AnsiString;
        L: Integer;
        Version: LongWord;
        VFPModeType: Integer;
        Mode: Integer;
begin
        // Get VeriFinger library name
        Name := VFGetStringParameter(VFP_NAME);
        // or
        Name := VFGetName;
        ShowMessage(Name);
        // Get VeriFinger library major version
        Version := VFGetDWordParameter(VFP_VERSION_HIGH);
        // or
        Version := VFGetVersionHigh;
        ShowMessage(Format('Version: %u.%u', [LoWord(Version),
        HiWord(Version)]));
        // Determine parameter VFP_MODE type
        VFPModeType := VFGetParameterType(VFP_MODE);
        // returned value: VF_TYPE_INT
        // Get integer parameter VFP_MODE value
        Mode := VFGetIntParameter(VFP_MODE);
        ShowMessage(Format('Mode: %d', [Mode]));
        // Set VFP_MODE parameter to VF_MODE_DIGITALPERSONA_URU
        VFSetIntParameter(VFP_MODE, VF_MODE_DIGITALPERSONA_URU);
end;
```

Visual Basic:

```
' Some application function/sub
Dim Name As String
Dim Version As Long
Dim Mode As Variant
' Get VeriFinger library name
Name = VFGetName
MsgBox Name
' Get VeriFinger library major version
Version = VFGetVersionHigh
MsgBox "Version: " & CStr((Version And &HFFFF0000) /
        &H10000) & "." & CStr(Version And 65535)
' Get parameter VFP_MODE value
VFGetParameter VFP_MODE, Mode, VF_DEFAULT_CONTEXT
MsgBox "Mode: " & CLng(Mode)
VFSetParameter VFP_MODE, VF_MODE_DIGITALPERSONA_URU, VF_DEFAULT_CONTEXT
```

Visual Basic .Net:

```
' Some application function/sub
Dim Name As String
Dim Version As Long
Dim Mode As Object
' Get VeriFinger library name
Name = VFGetName()
MsgBox(Name)
' Get VeriFinger library major version
Version = VFGetVersionHigh()
MsgBox("Version: " & CStr((Version And &HFFFF0000)
        / &H10000) & "." & CStr(Version And 65535))
' Get parameter VFP_MODE value
VFGetParameter(VFP_MODE, Mode, VF_DEFAULT_CONTEXT)
MsgBox("Mode: " & CLng(Mode))
VFSetParameter(VFP_MODE, VF_MODE_DIGITALPERSONA_URU, VF_DEFAULT_CONTEXT)
```

Java:

```
// Some application function (exception handling code omitted)
int ptype = VeriFingerWrapper.getParameterType(VFP_NAME);
System.out.println("VFP_NAME parameter type = " + Integer.toString(ptype));
```

8.2.7. 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.

8.2.7.1. 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 VeriFinger specific parameters.

C:

```
#define VF_IMAGE_RESOLUTION 500
#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
);
```

Delphi:

```
const
VF_IMAGE_RESOLUTION = 500;
VF_FEATURES_RESOLUTION = 500;
VF_MAX_FEATURES_SIZE = 10000;
function VFExtract
(
        Width,
        Height: Integer;
        Image: PByte;
        Resolution: Integer;
        Features: PByte;
        var Size: LongWord;
        Context: HVFCONTEXT
): Integer; stdcall;
```

Visual Basic:

```
Public Const VF_IMAGE_RESOLUTION = 500
Public Const VF_MIN_IMAGE_RESOLUTION = 50
Public Const VF_MAX_IMAGE_RESOLUTION = 5000
Public Const VF_MIN_IMAGE_DIMENSION = 16
Public Const VF_MAX_IMAGE_DIMENSION = 2048
Public Const VF_MAX_FEATURES_SIZE = 10000
Public Declare Function VFExtract Lib "VFVBP42.dll" Alias "VBVFExtract"
(
        ByVal Width As Long,
        ByVal Height As Long,
        Image As Variant,
        ByVal resolution As Long,
        Features As Variant,
        ByRef size As Long,
        ByVal context As Long
) As Long
```

Visual Basic .Net:

```
Public Const VF_IMAGE_RESOLUTION As Integer = 500
Public Const VF_MIN_IMAGE_RESOLUTION As Integer = 50
Public Const VF_MAX_IMAGE_RESOLUTION As Integer = 5000
Public Const VF_MIN_IMAGE_DIMENSION As Integer = 16
Public Const VF_MAX_IMAGE_DIMENSION As Integer = 2048
Public Const VF_MAX_FEATURES_SIZE As Integer = 10000
' Features extraction function
Public Declare Function VFExtract Lib "VFVBP42.dll" Alias "VBVFExtract"
(
        ByVal width As Integer,
        ByVal height As Integer,
        ByRef Image As Object,
        ByVal resolution As Integer,
        ByRef features As Object,
        ByRef size As Integer,
        ByVal context As Integer
) As Integer
```

Java:

```
public static final int VF_IMAGE_RESOLUTION= 500;
public static final int VF_MIN_IMAGE_RESOLUTION= 50;
public static final int VF_MAX_IMAGE_RESOLUTION= 5000;
public static final int VF_MIN_IMAGE_DIMENSION= 16;
public static final int VF_MAX_IMAGE_DIMENSION= 2048;
public static final int VF_MAX_FEATURES_SIZE= 10000;
// Features extraction function
public static native VeriFingerExtractionResult VFExtract
(
        int width,
        int height,
        byte[] image,
        int resolution,
        byte[] features,
        int context
) throws VeriFingerException, IllegalArgumentException,
                               Exception, ClassNotFoundException;
```

C#:

```
public const int imageResolution = 500;
public const int minImageResolution = 50;
public const int maxImageResolution = 5000;
public const int minImageDimention = 16;
public const int maxImageDimention = 2048;
// Features extraction function
public int Extract
(
        int width,
        int height,
        byte[] image,
        int resolution,
        out byte[] features
) throws VeriFingerException;
```

Parameters:

	width, Width	Fingerprint image width
	height, Height	Fingerprint image height
[in/out]	image, Image	Fingerprint image to extract features from. After execution of the function contains binarized or skeletonized image (see Parameters)
	resolution, Resolution	Resolution of the fingerprint image (in dots per inch)
[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 (in Visual Basic - `VF_DEFAULT_CONTEXT`)

Return values: If VeriFinger library is not registered returns `VFE_NOT_REGISTERED`.If context is null and VeriFinger 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.VeriFinger Java wrapper extraction function returns object (`VeriFingerExtractionResult` type) that contains size of the features in bytes and `VFE_LOW_QUALITY_IMAGE` warning flag.VeriFinger Java wrapper throws exception if error occurs.VeriFinger C# wrapper extraction function returns VeriFingerException.Ok, VeriFingerException.LowQualityImage. VeriFinger C# wrapper throws excepiton if error occurs.

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);
}
```

Delphi:

```
// Extraction function
var
        Width, Height: Integer;
        Image: PByte;
        Resolution: Integer;
        Features: array[0.. VF_MAX_FEATURES_SIZE - 1] of Byte;
        Size: LongWord;
begin
       // Load the image from file or get the image from scanner
       // ...
       VFExtract(Width, Height, Image, Resolution, @Features[0], Size, nil);
end;
```

Visual Basic:

```
' Extraction function
Dim Width as Long
Dim Height as Long
Dim Image() as Byte
Dim Resolution as Long
Dim Features(VF_MAX_FEATURES_SIZE) as Byte
Dim Size as Long
Dim ErrCode as Long
' Load the image from file or get the image from scanner
' ...
ErrCode = VFExtract(Width, Height, Image, Resolution, Features, Size,
                                                        VF_DEFAULT_CONTEXT)
```

Visual Basic .Net:

```
' Extraction function
Dim Width as Integer
Dim Height as Integer
Dim Image() as Byte
Dim Resolution as Integer
Dim Features(VF_MAX_FEATURES_SIZE) as Byte
Dim Size as Integer
Dim ErrCode as Integer
' Load the image from file or get the image from scanner
' ...
ErrCode = VFExtract(Width, Height, Image, Resolution, Features, Size,
                                                        VF_DEFAULT_CONTEXT)
```

Java:

```
// Extraction function (exception handling code omitted)
{
        int width, height;
        byte[] image;
        int resolution;
        byte[] features =
        new byte[VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
        // Load the image from file or get the image from scanner
        // ...
        VeriFingerExtractionResult r = VeriFingerWrapper.VFExtract(width,
                height, image, resolution, features, VF_DEFAULT_CONTEXT);
}
```

C#:

```
// Extraction function (exception handling code omitted)
        int width, height;
        byte[] image;
        int resolution;
        byte[] features;
        // Load the image from file or get the image from scanner
        // ...
        int ret = veriFinger.Extract(width, height, image,
                                        resolution, out features);
```

8.2.8. Features generalization

You can use features generalization to increase quality of the recognition. Generalization combines several feature collections to one collection, performs feature validation and noisy feature removal. You can use features generalization during enrollment. To obtain features 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.

8.2.8.1. 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 VeriFinger specific parameters.

C:

```
#define VF_GENERALIZE_COUNT 3
INT VFINGER_API VFGeneralize
(
        INT count,
        const BYTE * const * genFeatures,
        BYTE * features,
        DWORD * size,
        HVFCONTEXT context
);
```

Delphi:

```
type PPByte = ^PByte;
function VFGeneralize
(
        Count: Integer;
        GenFeatures: PPByte;
        Features: PByte;
        var Size: LongWord;
        Context: HVFCONTEXT
): Integer; stdcall;
```

Visual Basic:

```
Public Const VF_GENERALIZE_COUNT = 3
Public Declare Function VFGeneralize
Lib "VFVBP42.dll" Alias "VBVFGeneralize"
(
        ByVal count As Long,
        gen_features As Variant,
        features As Variant,
        ByRef Size As Long,
        ByVal context As Long
) As Long
```

Visual Basic .Net:

```
Public Const VF_GENERALIZE_COUNT As Integer = 3
' Features generalization function
Public Declare Function VFGeneralize
Lib "VFVBP42.dll" Alias "VBVFGeneralize"
(
        ByVal count As Integer,
        ByRef gen_features As Object,
        ByRef features As Object,
        ByRef size As Integer,
        ByVal context As Integer
) As Integer
```

Java:

```
public static final int VF_GENERALIZE_COUNT = 3;
public static native VeriFingerGeneralizationResult VFGeneralize
(
        int count,
        byte[][] gen_features,
        byte[] features,
        int context
) throws VeriFingerException, Exception;
```

C#:

```
public const int geheralizeCount = 3;
public byte[] Generalize
(
        int count,
        byte[][] genFeatures
) throws VeriFingerException;
```

Parameters:

	count, Count	Count of features collections to generalize
[in]	genFeatures, 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 (in Visual Basic - `VF_DEFAULT_CONTEXT`)

Return values: If VeriFinger library is not registered returns `VFE_NOT_REGISTERED`.If context is null and VeriFinger 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.VeriFinger Java wrapper generalization function returns object (`VeriFingerGeneralizationResult` type) that contains size of the features in bytes and index of features collection on which base features generalization has been performed.VeriFinger Java wrapper throws exception if error occurs.VeriFinger C# wrapper generalization function returns features array. VeriFinger C# wrapper throws VeriFingerException if error occurs.

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");
}
```

Delphi:

```
var
	Feats: array[0..2] of PByte;
	Features: array[0..VF_MAX_FEATURES_SIZE - 1] of Byte;
	Size: LongWord;
```

```
begin
        Feats[0] := {obtain first fingerprint features};
        Feats[1] := {obtain second fingerprint features};
        Feats[2] := {obtain third fingerprint features};
        if VFSucceeded(VFGeneralize(3, @Feats[0], @Features[0], Size, nil))
        then ShowMessage('Generalization succeeded')
        else ShowMessage('Generalization failed');
end;
```

Visual Basic:

```
Dim Feats(VF_GENERALIZE_COUNT-1) as Variant ' will hold 3 arrays of
'fingerprint features
Dim Features(VF_MAX_FEATURES_SIZE) as Byte
Dim Size as Long
Feats(0) = ' obtain first fingerprint features
Feats(1) = ' obtain second fingerprint features
Feats(2) = ' obtain third fingerprint features
if VFSucceeded(VFGeneralize(VF_GENERALIZE_COUNT, Feats, Features,
                                        Size, VF_DEFAULT_CONTEXT)) then
        MsgBox "Generalization succeeded"
else
        MsgBox "Generalization failed"
End if
```

Visual Basic .Net:

```
Dim Feats(VF_GENERALIZE_COUNT - 1) As Object
' will hold 3 arrays of fingerprint features
Dim Features(VF_MAX_FEATURES_SIZE) As Byte
Dim Size As Integer
Feats(0) = ' obtain first fingerprint features
Feats(1) = ' obtain second fingerprint features
Feats(2) = ' obtain third fingerprint features
If VFSucceeded(VFGeneralize(VF_GENERALIZE_COUNT, Feats, Features, Size,
                                                VF_DEFAULT_CONTEXT)) Then
        MsgBox("Generalization succeeded")
Else
        MsgBox("Generalization failed")
End If
```

Java:

```
// Generalization function
byte[][] gen_features = new byte[VeriFingerWrapper.VF_GENERALIZE_COUNT]
                                [VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
// Obtain features: gen_features[0], gen_features[1], gen_features[2]
byte[] features = new byte[VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
VeriFingerGeneralizationResult res = null;
try {
        res = VeriFingerWrapper.VFGeneralize(
                                VeriFingerWrapper.VF_GENERALIZE_COUNT,
                                gen_features, features, VF_DEFAULT_CONTEXT);
} catch (Exception e) {
// show message "Generalization failed!"
}
```

C#:

```
// Generalization function
byte[][] genFeatures = new byte[VeriFinger.geheralizeCount][];
// Obtain features: genFeatures[0], genFeatures[1], genFeatures[2]
byte[] featuresGeneralized;
try
{
        featuresGeneralized = veriFinger.Generalize(
                                VeriFinger.geheralizeCount, genFeatures);
}catch(VeriFingerException ex)
{
        if(ex.ErrorCode == VeriFingerException.Failed)
        {
        // show message "Features collection cannot be generalized"
        }
        else//error
        {
                MessageBox.Show(ex.Message);
        }
}
```

8.2.9. Verification

You can use verification to determine 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.

8.2.9.1. VFVerify function

Performs verification of two feature collections in the specified context.

Pass pointer to matching details structure that will receive details of features collections matching (set 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 VeriFinger specific parameters. For more information see Parameters.

C:

```
INT VFINGER_API VFVerify
```

```
(
        const BYTE * features1,
        const BYTE * features2,
        VFMatchDetails * md,
        HVFCONTEXT context
);
```

Delphi:

```
function VFVerify
(
        Features1,
        Features2: PByte;
        MD: PVFMatchDetails;
        Context: HVFCONTEXT
): Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFVerify Lib "VFVBP42.dll" Alias "VBVFVerify"
(
        features1 As Variant,
        features2 As Variant,
        md As Any,
        ByVal context As Long
) As Long
```

Visual Basic .Net:

```
Public Declare Function VFVerify Lib "VFVBP42.dll" Alias "VBVFVerify"
(
        ByRef features1 As Object,
        ByRef features2 As Object,
        ByRef md As VFMATCHDETAILS,
        ByVal context As Integer
) As Integer

Public Declare Function VFVerify Lib "VFVBP42.dll" Alias "VBVFVerify"
(
        ByRef features1 As Object,
        ByRef features2 As Object,
        ByRef md As VFMATCHDETAILSEX,
        ByVal context As Integer
) As Integer
```

Java:

```
public static native VeriFingerMatchDetails VFVerify
(
        byte[] features1,
        byte[] features2,
        boolean extended_match_info,
        int context
) throws VeriFingerException, Exception;
```

C#:

```
public int Verify
(
        byte[] features1,
        byte[] features2,
        MatchDetails md
) throws VeriFingerException;
```

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 (in Visual Basic - `VF_DEFAULT_CONTEXT`)

Return values: If VeriFinger library is not registered returns `VFE_NOT_REGISTERED`.If context is `null` and VeriFinger 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`.VeriFinger Java wrapper verification function returns object (`VeriFingerMatchDetails` type) that contains similarity and other information generated by matching algorithm.VeriFinger Java wrapper throws exception if error occurs.VeriFinger Java wrapper does not throw exception if features collections similarity is not high enough (see `VFP_MATCHING_THRESHOLD` in Parameters) therefore similarity must be evaluated manualy.VeriFinger C# wrapper returns VeriFingerException.Ok or VeriFingerException.Failed. Otherwise throws VeriFingerException exception.

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);
}
```

Delphi:

```
// Verification function
var
        Features1, Features2: PByte;
        MD: TVFMatchDetails;
        Result: Boolean;
begin
        Features1 := {obtain first fingerprint features};
        Features2 := {obtain second fingerprint features};
        MD.Size := SizeOf(MD);
        Result := VFSucceeded(VFVerify(Features1, Features2, @MD, nil));
        if Result then
                ShowMessage(Format('Same finger. Similarity: %d',
                                                        MD.Similarity))
        else
                ShowMessage('Different fingers. Similarity: %d',
                                                        MD.Similarity);
end;
```

Visual Basic:

```
' Verification function
Dim Features1(VF_MAX_FEATURES_SIZE) as Byte
Dim Features2(VF_MAX_FEATURES_SIZE) as Byte
Dim md as VFMatchDetails
Dim Result as Boolean
Features1 = ' obtain first fingerprint features
Features2 = ' obtain second fingerprint features
md.Size = VF_MATCHDETAILS_SIZE
Result = VFSucceeded(VFVerify(Features1, Features2, md, VF_DEFAULT_CONTEXT))
if Result then
        MsgBox "Same finger. Similarity: " & CStr(md.Similarity)
else
        MsgBox "Different fingers. Similarity: " & CStr(md.Similarity)
End if
```

Visual Basic .Net:

```
' Verification function
Dim Features1(VF_MAX_FEATURES_SIZE) As Byte
Dim Features2(VF_MAX_FEATURES_SIZE) As Byte
Dim md As VFMATCHDETAILS
Dim Result As Boolean
Features1 = ' obtain first fingerprint features
Features2 = ' obtain second fingerprint features
md.size = VF_MATCHDETAILS_SIZE
Result = VFSucceeded(VFVerify(Features1, Features2, md, VF_DEFAULT_CONTEXT))
If Result Then
        MsgBox("Same finger. Similarity: " & CStr(md.similarity))
Else
        MsgBox("Different fingers. Similarity: " & CStr(md.similarity))
End If
```

Java:

```
// Verification function
byte[] features1 = new byte [VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
byte[] features2 = new byte [VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
// obtain features (read from DB or extract from fingerprint image)
VeriFingerMatchDetails res = null;
try {
        res = VeriFingerWrapper.VFVerify(features1, features2, true,
                                                        VF_DEFAULT_CONTEXT);
} catch (Exception e) {
        // error handler
}
if (res != null)
{
        System.out.println("Similarity = " + Integer.toString(
                                                        res.similarity));
}
```

C#:

```
// Verification function
byte[] features1;
byte[] features2;
// obtain features (read from DB or extract from fingerprint image)
VeriFinger.MatchDetails res = new VeriFinger.MatchDetails();
int ret;
try
{
        ret = veriFinger.Verify(features1, features2, res);
}catch(VeriFingerException ex)
{}
if(ret != VeriFingerException.Failed)
{
        //get similarity
        //res.similarity;
}
```

8.2.10. 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 * testFeatures;
        BYTE * sampleFeatures;
        BOOL found;
        testFeatures = /*obtain features of fingerprint to identify*/;
        VFIdentifyStart(testFeatures, NULL);
        found = FALSE;
        for (/* walk through database */)
        {
                sampleFeatures = /*some features from the database*/;
                if(VFSucceeded(VFIdentifyNext(sampleFeatures, NULL, NULL)))
                {
                        found = TRUE;
                        break;
                }
        }
        VFIdentifyEnd(NULL);
        if (found)
                printf("Fingerprint found in the database");
        else
                printf("Fingerprint not found in the database");
}
```

Delphi:

```
// Identification function
var
	TestFearures: PByte;
	SampleFeatures: PByte;
	Found: Boolean;
begin
	TestFeatures := {obtain features of fingerprint to identify};
	VFIdentifyStart(TestFeatures, nil);
	Found := False;
	for { walk through database } do
	begin
		SampleFeatures := {some features from the database};
		if VFSucceeded(VFIdentifyNext(SampleFeatures, nil, nil))then
		begin
			Found := True;
			Break;
		end;
	end;
	VFIdentifyEnd(nil);
	if Found then
		ShowMessage('Fingerprint found in the database')
	else
		ShowMessage('Fingerprint not found in the database');
end;
```

Visual Basic:

```
' Identification function
Dim TestFearures as Variant
Dim SampleFeatures as Variant
Dim Found as Boolean
TestFeatures = ' obtain features of fingerprint to identify
VFIdentifyStart TestFeatures, VF_DEFAULT_CONTEXT
Found = False
for ' walk through database
SampleFeatures = ' some features from the database
if VFSucceeded(VFIdentifyNext(SampleFeatures, 0, VF_DEFAULT_CONTEXT)) then
	Found = True
	Exit For
End if
Next ' fingerprint
VFIdentifyEnd VF_DEFAULT_CONTEXT
if Found then
	MsgBox "Fingerprint found in the database"
else
	MsgBox "Fingerprint not found in the database"
End if
```

Visual Basic .Net:

```
' Identification function
```

```
Dim TestFearures As Object
Dim SampleFeatures As Object
Dim Found As Boolean
TestFeatures = ' obtain features of fingerprint to identify
VFIdentifyStart(TestFeatures, VF_DEFAULT_CONTEXT)
Found = False
for ' walk through database
SampleFeatures = ' some features from the database
If VFSucceeded(VFIdentifyNext(SampleFeatures, 0, VF_DEFAULT_CONTEXT)) Then
        Found = True
        Exit For
End If
Next ' fingerprint
VFIdentifyEnd(VF_DEFAULT_CONTEXT)
If Found Then
        MsgBox("Fingerprint found in the database")
Else
        MsgBox("Fingerprint not found in the database")
End If
```

Java:

```
// Identification function (exception handling code omitted)
byte[] test_features = new byte [VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
byte[] sample_features = new byte [VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
boolean found = false;
test_features = /*obtain features of fingerprint to identify*/;
VeriFingerWrapper.VFIdentifyStart(test_features, VF_DEFAULT_CONTEXT);
for (/* walk through database */)
{
        sample_features = /*some features from the database*/;
        VeriFingerMatchDetails res = VeriFingerWrapper.VFIdentifyNext(
                                sample_features, true, VF_DEFAULT_CONTEXT)))
        if (res.similarity >= /* desired result*/) {
                found = TRUE;
                break;
        }
}
VeriFingerWrapper.VFIdentifyEnd(VF_DEFAULT_CONTEXT);
if (found)
        System.out.printf("Fingerprint found in the database");
else
        System.out.printf("Fingerprint not found in the database");
```

C#:

```
// Identification function (exception handling code omitted)
byte[] testFeatures;
byte[] sampleFeatures;
bool found = false;
testFeatures = /*obtain features of fingerprint to identify*/;
veriFinger.IdentifyStart(testFeatures);
for (/* walk through database */)
```

```
{
        sampleFeatures = /*some features from the database*/;
        int ret = veriFinger.IdentifyNext(sampleFeatures, matchDetails);
        if(ret == VeriFingerException.Ok)
        {
                if(matchDetails.similarity >= /* desired result*/)
                {
                        found = TRUE;
                        break;
                }
        }

}
veriFinger.IdentifyEnd();
if(found)
{
        //Fingerprint found in the database"
}
else
{
        //Fingerprint not found in the database
}
```

8.2.10.1. VFIdentifyStart function

Starts identification with specified test features in specified context.

This function uses features matching and VeriFinger specific parameters. For more information see Parameters.

C:

```
INT VFINGER_API VFIdentifyStart
(
        const BYTE * testFeatures,
        HVFCONTEXT context
);
```

Delphi:

```
function VFIdentifyStart
(
        TestFeatures: PByte;
        Context: HVFCONTEXT
): Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFIdentifyStart Lib "VFVBP42.dll"
Alias "VBVFIdentifyStart"
(
        test_features As Variant,
        ByVal context As Long
) As Long
```

Visual Basic .Net:

```
Public Declare Function VFIdentifyStart Lib "VFVBP42.dll"
Alias "VBVFIdentifyStart"
(
        ByRef test_features As Object,
        ByVal context As Integer
) As Integer
```

Java:

```
public static native void VFIdentifyStart
(
        byte[] test_features,
        int context
) throws VeriFingerException, Exception, IllegalArgumentException;
```

C#:

```
public void IdentifyStart(byte[] testFeatures) throws VeriFingerException;
```

Parameters:

[in]	testFeatures, TestFeatures	Test features
	context, Context	Context to start identification in `Null` for default context (in Visual Basic - `VF_DEFAULT_CONTEXT`)

Return values: If VeriFinger library is not registered returns `VFE_NOT_REGISTERED`.If context is null and VeriFinger 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`.VeriFinger Java wrapper throws exception if error occurs.VeriFinger C# wrapper throws VeriFingerException if error occurs.

8.2.10.2. 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 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 VeriFinger specific parameters. For more information see Parameters.

C:

```
INT VFINGER_API VFIdentifyNext
(
        const BYTE * sampleFeatures,
        VFMATCHDETAILS * md,
        HVFCONTEXT context
);
```

Delphi:

```
function VFIdentifyNext
(
        SampleFeatures: PByte;
        MD: PVFMatchDetails;
        Context: HVFCONTEXT
): Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFIdentifyNext Lib "VFVBP42.dll"
Alias "VBVFIdentifyNext"
(
        sample_features As Variant,
        md As Any,
        ByVal context As Long
) As Long
```

Visual Basic .Net:

```
Public Declare Function VFIdentifyNext Lib "VFVBP42.dll"
Alias "VBVFIdentifyNext"
(
        ByRef sample_features As Object,
        ByRef md As VFMATCHDETAILS,
        ByVal context As Integer
) As Integer
```

Java:

```
public static native VeriFingerMatchDetails VFIdentifyNext
(
        byte[] sample_features,
        boolean extendedMatchInfo,
        int context
) throws VeriFingerException, Exception,
                        IllegalArgumentException, ClassNotFoundException;
```

C#:

```
public int IdentifyNext
(
        byte[] sampleFeatures,
        MatchDetails md
) throws VeriFingerException;
```

Parameters:

[in]	sampleFeatures, 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 (in Visual Basic - `VF_DEFAULT_CONTEXT`)

Return values: If VeriFinger library is not registered returns `VFE_NOT_REGISTERED`.If context is null and VeriFinger 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`.VeriFinger Java wrapper verification function returns object (VeriFingerMatchDetails type) that contains similarity and other information generated by matching algorithm.VeriFinger Java wrapper throws exception if error occurs.VeriFinger Java wrapper does not throw exception if features collections similarity is not high enough (see `VFP_MATCHING_THRESHOLD` in Parameters) therefore similarity must be evaluated manualy.VeriFinger C# wrapper function returns VeriFingerException.Ok, VeriFingerException.Failed. Otherwise throws VeriFingerException.

8.2.10.3. VFIdentifyEnd function

Ends the identification started with VFIdentifyStart function in specified context.

C:

```
INT VFINGER_API VFIdentifyEnd(HVFCONTEXT context);
```

Delphi:

```
function VFIdentifyEnd(Context: HVFCONTEXT): Integer; stdcall;
```

Visual Basic:

```
Public Declare Function VFIdentifyEnd Lib "VFVBP42.dll"
Alias "VBVFIdentifyEnd"
(
        ByVal context As Long
) As Long
```

Visual Basic .Net:

```
Public Declare Function VFIdentifyEnd Lib "VFVBP42.dll"
Alias "VBVFIdentifyEnd"
(
        ByVal context As Integer
) As Integer
```

Java:

```
public static native void VFIdentifyEnd(int context)
throws VeriFingerException, Exception, IllegalArgumentException;
```

C#:

```
public int IdentifyEnd() throws VeriFingerException;
```

Parameters:

	context, Context	Context to end identification in. Null for default context (in Visual Basic - `VF_DEFAULT_CONTEXT`)

Return values: If VeriFinger library is not registered returns `VFE_NOT_REGISTERED`.If context is null and VeriFinger library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is not started returns `VFE_INVALID_MODE`.VeriFinger Java wrapper throws exception if error occurs.VeriFinger C# wrapper throws VeriFingerException if error occurs.

8.2.11. Matching threshold and similarity

VeriFinger 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 VeriFinger library; for C they are implemented in `VFingerX.h` and `VFingerX.cpp` files, for Delphi in `VFinger.pas` module.

C:

```
double VFMatchingThresholdToFAR(INT th);
INT VFFARToMatchingThreshold(double f);
```

Delphi:

```
function VFMatchingThresholdToFAR(Th: Integer): Double;
function VFFARToMatchingThreshold(F: Double): Integer;
```

C#:

```
public double MatchingThresholdToFAR(int th);
public int FARToMatchingThreshold(double f);
```

8.2.12. 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:

```
typedef struct _VFMatchDetails
{
        DWORD Size;
        INT Similarity;
        INT Rotation;
        INT TranslatonX;
        INT TranslationT;
} VFMatchDetails;

#define VF_MAX_MATCHED_MINUTIA_COUNT 1024

typedef struct _VFMatchedMinutiae
{
        INT Count;
        SHORT Test[VF_MAX_MATCHED_MINUTIA_COUNT];
        SHORT Sample[VF_MAX_MATCHED_MINUTIA_COUNT];
} VFMatchedMinutiae;

typedef struct _VFMatchDetailsEx
{
        DWORD Size;
        INT Similarity;
        INT Rotation;
        INT TranslationX;
        INT TranslationY;
        VFMatchedMinutiae MM;
} VFMatchDetailsEx;
```

Delphi:

```
type
        PVFMatchDetails = ^TVFMatchDetails;
        TVFMatchDetails = record
        Size: LongWord;
        Similarity: Integer;
        Rotation: Integer;
        TranslationX: Integer;
        TranslationY: Integer;
end;

const VF_MAX_MATCHED_MINUTIA_COUNT = 1024;

type PVFMatchedMinutiae = ^TVFMatchedMinutiae;
TVFMatchedMinutiae = record
        Count: Integer;
        Test: array[0..VF_MAX_MATCHED_MINUTIA_COUNT - 1] of SmallInt;
        Sample: array[0..VF_MAX_MATCHED_MINUTIA_COUNT - 1] of SmallInt;
end;

PVFMatchDetailsEx = ^TVFMatchDetailsEx;
TVFMatchDetailsEx = record
        Size: LongWord;
        Similarity: Integer;
        Rotation: Integer;
        TranslationX: Integer;
        TranslationY: Integer;
        MM: TVFMatchedMinutiae;
end;
```

Visual Basic:

```
Public Const VF_MAX_MINUTIA_COUNT = 1024
Public Const VF_MATCHDETAILS_SIZE = 20
Public Const VF_MATCHDETAILSEX_SIZE = 4120

Public Type VFMATCHDETAILS
        Size As Long
        similarity As Long
        rotation As Long
        trans_x As Long
        trans_y As Long
End Type

Public Type VFMATCHDETAILSEX
        Size As Long
        similarity As Long
        rotation As Long
        trans_x As Long
        trans_y As Long
        mm_count As Long
        mm(VF_MAX_MINUTIA_COUNT - 1, 1) As Integer
End Type
```

Visual Basic .Net:

```
Public Const VF_MAX_MINUTIA_COUNT As Integer = 1024
Public Const VF_MATCHDETAILS_SIZE As Integer = 20
Public Const VF_MATCHDETAILSEX_SIZE As Integer = 4120

Public Structure VFMATCHDETAILS
        Dim size As Integer ' DWORD
        Dim similarity As Integer ' INT
        Dim rotation As Integer ' INT
        Dim trans_x As Integer ' INT
        Dim trans_y As Integer ' INT
End Structure

<StructLayout(LayoutKind.Sequential)> _
Public Structure VFMATCHDETAILSEX
        Dim size As Integer ' DWORD
        Dim similarity As Integer ' INT
        Dim rotation As Integer ' INT
        Dim trans_x As Integer ' INT
        Dim trans_y As Integer ' INT
        Dim mm_count As Integer ' INT
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_MINUTIA_COUNT * 2), _
        VBFixedArray(VF_MAX_MINUTIA_COUNT * 2 - 1)> _
        Dim mm() As Integer ' SHORT
        ' access data: mm(index*2 + 0 or 1)
        Public Sub Initialize()
                ReDim mm(VF_MAX_MINUTIA_COUNT * 2 - 1)
        End Sub
End Structure
```

Java:

```
public class VeriFingerWrapper {
        ...
        public static final int VF_MAX_MINUTIA_COUNT= 1024;
        ...
        public class VeriFingerMatchDetails {
        public int similarity;
        public int rotation;
        public int trans_x;
        public int trans_y;
        // extended information:
        public int mm_count;
        public short[][] mm;
        public VeriFingerMatchDetails()
        {
                mm = new short[2][VeriFingerWrapper.VF_MAX_MINUTIA_COUNT];
        }
}
```

C#:

```
public class MatchDetails
{
        public int similarity;
        public int rotation;
        public int translationX;
        public int translationT;

        public MatchDetails()
        {}
}
```

Members:

`Size`	Size of the structure. Set this member before calling a function
`Similarity`	Two features collections similarity value. The bigger is value the higher is similarity. See also Matching threshold and similarity
`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
`TranslationX`	Translation between two features collections along X axis
`TranslationY`	Translation between two features collections along Y axis

MM.Count	Count of minutiae common for two features collections in MM collection
MM.Test *MM.Sample*	Matched minutiae arrays (common minutiae for two features collections). First - index of minutia in test (first) features collection, second - index of minutia in sample (second) features collection. See also Features

Example:

C:

```
//Identification function
{
        BYTE * testFeatures;
        BYTE * sampleFeatures;
        VFMatchDetails md;
        //...
        VFIdentifyStart(testFeatures, NULL);
        md.Size = sizeof(md);
        for (...)
        {
                VFIdentifyNext(sampleFeatures, &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.Test[i], md.MM.Sample[i]);
}
```

Delphi:

```
//Identification function
var
        TestFeatures: PByte;
        SampleFeatures: PByte;
        MD: TVFMatchDetails;
begin
```

```
        //...
        VFIdentifyStart(TestFeatures, nil);
        MD.Size := SizeOf(MD);
        for ... do
        begin
                VFIdentifyNext(SampleFeatures, @MD, nil);
                ShowMessage(Format('Similarity: %d', [MD.Similarity]));
        end;
        VFIdentifyEnd(nil);
end;
// Verification function
var
        Features1, Features2: PByte;
        MD: TVFMatchDetailsEx;
        I: Integer;
begin
        // ...
        MD.Size := SizeOf(MD);
        VFVerify(Features1, Features2, PVFMatchDetails(@MD), nil);
        for I := 0 to MD.MMCount - 1 do
        ShowMessage(Format('Matched minutia %d: '
        'index in first features - %d; index in second features - %d',
        [I, MD.MM.Test[I], MD.MM.Sample[I]]));
end;
```

Visual Basic:

```
' Identification function
Dim test_features...
Dim sample_features...
Dim md as VFMATCHDETAILS
' ...
VFIdentifyStart test_features, VF_DEFAULT_CONTEXT
md.size = VF_MATCHDETAILS_SIZE
for ...
        VFIdentifyNext sample_features, md, VF_DEFAULT_CONTEXT
        Debug.Print "Similarity: ", CStr(md.similarity)
next...
VFIdentifyEnd(VF_DEFAULT_CONTEXT)
' Verification function
Dim features1...
Dim features2...
Dim md as VFMATCHDETAILSEX
Dim i as Long
' ...
md.size = VF_MATCHDETAILS_SIZE
VFVerify features1, features2, md, VF_DEFAULT_CONTEXT
For i = 0 to i = md.mm_count-1
        Debug.Print "Matched minutia " & CStr(i) & _
        ": index in first features - " & CStr(md.mm[0][i]) & _
        "; index in second features - " & CStr(md.mm[1][i]) & _
        vbNewLine
Next i
```

Visual Basic .Net:

```
' Identification function
Dim test_features...
Dim sample_features...
Dim md as VFMATCHDETAILS
' ...
VFIdentifyStart(test_features, VF_DEFAULT_CONTEXT)
md.size = VF_MATCHDETAILS_SIZE
for ...
        VFIdentifyNext(sample_features, md, VF_DEFAULT_CONTEXT)
        Debug.Write("Similarity: ", CStr(md.similarity))
next...
VFIdentifyEnd(VF_DEFAULT_CONTEXT)
' Verification function
Dim features1...
Dim features2...
Dim md as VFMATCHDETAILSEX
Dim i as Long
' ...
md.size = VF_MATCHDETAILSEX_SIZE
VFVerify(features1, features2, md, VF_DEFAULT_CONTEXT)
For i = 0 to i = md.mm_count-1
        Debug.Write("Matched minutia " & CStr(i) & _
        ": index in first features - " & CStr(md.mm[i*2 + 0]) & _
        "; index in second features - " & CStr(md.mm[i*2 + 1]) & _
        vbNewLine)
Next i
```

Java:

```
// Identification function (exception handling code omitted)
byte[] test_features = new byte [VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
byte[] sample_features = new byte [VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
test_features = /*obtain features of fingerprint to identify*/;
VeriFingerWrapper.VFIdentifyStart(test_features, VF_DEFAULT_CONTEXT);
for (/* walk through database */)
{
       sample_features = /*some features from the database*/;
       VeriFingerMatchDetails res = VeriFingerWrapper.VFIdentifyNext(
                                sample_features, true, VF_DEFAULT_CONTEXT)))
       System.out.println("Similarity: " + Integer.toString(res.similarity);
}
VeriFingerWrapper.VFIdentifyEnd(VF_DEFAULT_CONTEXT);
// Verification function
byte[] features1 = new byte [VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
byte[] features2 = new byte [VeriFingerWrapper.VF_MAX_FEATURES_SIZE];
// obtain features (read from DB or extract from fingerprint image)
VeriFingerMatchDetails res = null;
try {
        res = VeriFingerWrapper.VFVerify(features1, features2, true,
                                                       VF_DEFAULT_CONTEXT);
} catch (Exception e) {
// error handler
```

```
}
if (res != null)
{
        System.out.println("Similarity = " + Integer.toString(
                                                res.similarity));
        for (int i = 0; i < res.mm_count; ++i)
        System.out.println("Matched minutia " + Integer.toString(i) +
        ": index in first features - " + Integer.toString(res.mm[0][i]) +
        "; index in second features - " + Integer.toString(res.mm[1][i]));
}
```

8.2.13. 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 `VFFeatGetXxx` and `VFFeatSetXxx` functions to decompress and compress features. Also you may use `CVFFeatures` class in C (`VFFeatures.h` and `VFFeatures.cpp` files in sample application) and `TVFFeatures` class in Delphi (`VFFeatures.pas` module) 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);
```

Delphi:

```
TVFSingularPointType = (
        vfsptUnknown = 0,
        vfsptCore = 1,
        vfsptDoubleCore = 2,
        vfsptDelta = 3);

PVFSingularPoint = ^TVFSingularPoint;
TVFSingularPoint = record
        X: Integer;
        Y: Integer;
        T: TVFSingularPointType;
        Dummy1, Dummy2, Dummy3: Byte; // because enumeration have to be
        // 4 bytes
        D: Byte;
end;

TVFMinutiaType = (
        vfmtUnknown = 0,
        vfmtEnd = 1,
        vfmtBifurcation = 2);

PVFMinutia = ^TVFMinutia;
TVFMinutia = record
        X: Integer;
        Y: Integer;
        T: TVFMinutiaType;
        Dummy1, Dummy2, Dummy3: Byte; // because enumeration have to be
        // 4 bytes
        D: Byte;
        C: Byte;
        G: Byte;
end;

// Features compression
function VFFeatSet
(
        G: Byte;
        MCount: Integer; M: PVFMinutia;
        SPCount: Integer; SP: PVFSingularPoint;
        BOWidth, BOHeight: Integer; BO: PByte; Features: PByte
): Integer; stdcall;

// Features decompression
function VFFeatGetG(Features: PByte): Integer; stdcall;
function VFFeatGetMinutiaCount(Features: PByte): Integer; stdcall;
function VFFeatGetMinutiae(Features: PByte;
                                        M: PVFMinutia): Integer; stdcall;
function VFFeatGetSPCount(Features: PByte): Integer; stdcall;
function VFFeatGetSP(Features: PByte;
                                SP: PVFSingularPoint): Integer; stdcall;
function VFFeatGetBOSize(Features: PByte; var Width,
                                Height: Integer): Integer; stdcall;
function VFFeatGetBO(Features: PByte; BO: PByte): Integer; stdcall;
```

Visual Basic:

```
' Features structure definition
Public Const VF_MAX_MINUTIA_COUNT = 1024
Public Const VF_MAX_IMAGE_DIMENSION = 2048

Public Type VFMINUTIAE
        count As Long ' int
        X(VF_MAX_MINUTIA_COUNT - 1) As Long ' int
        Y(VF_MAX_MINUTIA_COUNT - 1) As Long ' int
        D(VF_MAX_MINUTIA_COUNT - 1) As Byte ' BYTE
        C(VF_MAX_MINUTIA_COUNT - 1) As Byte ' BYTE
End Type

Public Const VF_MAX_SINGULAR_POINT_COUNT = 64

Public Type VFSINGULARPOINTS
        count As Long ' int
        X(VF_MAX_SINGULAR_POINT_COUNT - 1) As Long ' int
        Y(VF_MAX_SINGULAR_POINT_COUNT - 1) As Long ' int
        T(VF_MAX_SINGULAR_POINT_COUNT - 1) As Byte ' BYTE
        D(VF_MAX_SINGULAR_POINT_COUNT - 1) As Byte ' BYTE
End Type

Public Const VF_BLOCK_SIZE = 16

Public Const VF_MAX_BLOCKED_ORIENTS_DIMENSION = (
                                VF_MAX_IMAGE_DIMENSION / VF_BLOCK_SIZE)
```

```
Public Type VFBLOCKEDORIENTS
        width As Long ' int
        height As Long ' int
        bits(VF_MAX_BLOCKED_ORIENTS_DIMENSION - 1,
                VF_MAX_BLOCKED_ORIENTS_DIMENSION - 1) As Byte ' BYTE
End Type

Public Type VFFEATURES
        G As Byte ' BYTE
        m As VFMINUTIAE
        sp As VFSINGULARPOINTS
        bo As VFBLOCKEDORIENTS
End Type

' Compression/decompression functions
Public Declare Function VFDecompressFeatures Lib "VFVBP42.dll"
Alias "VBVFDecompressFeatures"
(
        features As Variant,
        structure As VFFEATURES
) As Long
Public Declare Function VFCompressFeatures Lib "VFVBP42.dll"
Alias "VBVFCompressFeatures"
(
        structure As VFFEATURES,
        features As Variant,
        ByRef size As Long
) As Long
```

Visual Basic .Net:

```
' Features structure definition
Public Const VF_MAX_MINUTIA_COUNT As Integer = 1024
Public Const VF_MAX_IMAGE_DIMENSION As Integer = 2048
<StructLayout(LayoutKind.Sequential)> _
Public Structure VFMINUTIAE
        Dim count As Integer ' int
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_MINUTIA_COUNT), _
        VBFixedArray(VF_MAX_MINUTIA_COUNT - 1)> _  Dim X() As Integer ' int
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_MINUTIA_COUNT), _
        VBFixedArray(VF_MAX_MINUTIA_COUNT - 1)> _
        Dim Y() As Integer ' int
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_MINUTIA_COUNT), _
        VBFixedArray(VF_MAX_MINUTIA_COUNT - 1)> _
        Dim D() As Byte ' BYTE
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_MINUTIA_COUNT), _
        VBFixedArray(VF_MAX_MINUTIA_COUNT - 1)> _
        Dim C() As Byte ' BYTE
        Public Sub Initialize()
                ReDim X(VF_MAX_MINUTIA_COUNT - 1)
                ReDim Y(VF_MAX_MINUTIA_COUNT - 1)
                ReDim D(VF_MAX_MINUTIA_COUNT - 1)
                ReDim C(VF_MAX_MINUTIA_COUNT - 1)
        End Sub
End Structure
Public Const VF_MAX_SINGULAR_POINT_COUNT As Integer = 64
<StructLayout(LayoutKind.Sequential)> _
Public Structure VFSINGULARPOINTS
        Dim count As Integer ' int
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_SINGULAR_POINT_COUNT), _
        VBFixedArray(VF_MAX_SINGULAR_POINT_COUNT - 1)> _
        Dim X() As Integer ' int
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_SINGULAR_POINT_COUNT), _
        VBFixedArray(VF_MAX_SINGULAR_POINT_COUNT - 1)> _
        Dim Y() As Integer ' int
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_SINGULAR_POINT_COUNT), _
        VBFixedArray(VF_MAX_SINGULAR_POINT_COUNT - 1)> _
        Dim T() As Byte ' BYTE
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_SINGULAR_POINT_COUNT), _
        VBFixedArray(VF_MAX_SINGULAR_POINT_COUNT - 1)> _
        Dim D() As Byte ' BYTE
        Public Sub Initialize()
                ReDim X(VF_MAX_SINGULAR_POINT_COUNT - 1)
                ReDim Y(VF_MAX_SINGULAR_POINT_COUNT - 1)
                ReDim T(VF_MAX_SINGULAR_POINT_COUNT - 1)
                ReDim D(VF_MAX_SINGULAR_POINT_COUNT - 1)
        End Sub
End Structure
Public Const VF_BLOCK_SIZE As Integer = 16
Public Const VF_MAX_BLOCKED_ORIENTS_DIMENSION As Integer = (
                                VF_MAX_IMAGE_DIMENSION / VF_BLOCK_SIZE)
<StructLayout(LayoutKind.Sequential)> _
Public Structure VFBLOCKEDORIENTS
        Dim width As Integer ' int
        Dim height As Integer ' int
        <MarshalAs(UnmanagedType.ByValArray, _
        SizeConst:=VF_MAX_BLOCKED_ORIENTS_DIMENSION * _
        VF_MAX_BLOCKED_ORIENTS_DIMENSION), _
        VBFixedArray(VF_MAX_BLOCKED_ORIENTS_DIMENSION * _
        VF_MAX_BLOCKED_ORIENTS_DIMENSION - 1)> _
        Dim bits() As Byte ' BYTE
        ' access data: bits(y*VF_MAX_BLOCKED_ORIENTS_DIMENSION + x)
        Public Sub Initialize()
                ReDim bits(VF_MAX_BLOCKED_ORIENTS_DIMENSION * _
                VF_MAX_BLOCKED_ORIENTS_DIMENSION - 1)
        End Sub
End Structure
<StructLayout(LayoutKind.Sequential)> _
Public Structure VFFEATURES
        Dim G As Byte ' BYTE
        <MarshalAs(UnmanagedType.Struct)>_
        Dim m As VFMINUTIAE
        <MarshalAs(UnmanagedType.Struct)> _
        Dim sp As VFSINGULARPOINTS
        <MarshalAs(UnmanagedType.Struct)> _
        Dim bo As VFBLOCKEDORIENTS
        Public Sub Initialize()
                m.Initialize()
                sp.Initialize()
                bo.Initialize()
        End Sub
End Structure
' Compression/decompression functions
Public Declare Function VFDecompressFeatures Lib "VFVBP42.dll"
Alias "VBVFDecompressFeatures"
(
        ByRef features As Object,
        ByRef featuresstructure As VFFEATURES
) As Integer
Public Declare Function VFCompressFeatures Lib "VFVBP42.dll"
Alias "VBVFCompressFeatures"
(
        ByRef featuresstructure As VFFEATURES,
        ByRef features As Object,
        ByRef size As Integer
) As Integer
```

Java:

```
public class VeriFingerWrapper {
        ...
        public static final int VF_MAX_IMAGE_DIMENSION= 2048;
        public static final int VF_MAX_MINUTIA_COUNT= 1024;
        public static final int VF_MAX_SINGULAR_POINT_COUNT= 64;
        public static final int VF_BLOCK_SIZE= 16;
        public static final int VF_MAX_BLOCKED_ORIENTS_DIMENSION = (
                                VF_MAX_IMAGE_DIMENSION / VF_BLOCK_SIZE);
        public static final int VF_MAX_FEATURES_SIZE= 10000;
        public static native VeriFingerFeatures VFDecompressFeatures(
                byte[] features) throws VeriFingerException, Exception;
        public static native int VFCompressFeatures(VeriFingerFeatures f,
                byte[] features) throws VeriFingerException, Exception;
        ...
        public class VeriFingerFeatures implements Serializable {
        public byte g;
        public int m_count; // minutiae
        public int[] m_x;
        public int[] m_y;
        public byte[] m_d;
        public byte[] m_c;
        public int sp_count; // singular points
        public int[] sp_x;
        public int[] sp_y;
        public byte[] sp_t;
        public byte[] sp_d;
        public int bo_width; // blocked orientations
        public int bo_height;
        public byte[][] bo_bits;
        public VeriFingerFeatures()
        {
             m_x = new int[VeriFingerWrapper.VF_MAX_MINUTIA_COUNT];
             m_y = new int[VeriFingerWrapper.VF_MAX_MINUTIA_COUNT];
             m_d = new byte[VeriFingerWrapper.VF_MAX_MINUTIA_COUNT];
             m_c = new byte[VeriFingerWrapper.VF_MAX_MINUTIA_COUNT];
             sp_x = new int[VeriFingerWrapper.VF_MAX_SINGULAR_POINT_COUNT];
             sp_y = new int[VeriFingerWrapper.VF_MAX_SINGULAR_POINT_COUNT];
             sp_t = new byte[VeriFingerWrapper.VF_MAX_SINGULAR_POINT_COUNT];
             sp_d = new byte[VeriFingerWrapper.VF_MAX_SINGULAR_POINT_COUNT];
             int dim = VeriFingerWrapper.VF_MAX_BLOCKED_ORIENTS_DIMENSION;
             bo_bits = new byte[dim][dim];
        }
};
```

C#:

```
public enum VFSingularPointType: int
{
        vfsptUnknown = 0,
        vfsptCore = 1,
        vfsptDoubleCore = 2,
        vfsptDelta = 3
}

public struct SingularPoint
{
        public int X;
        public int Y;
        public VFSingularPointType T;
        public byte D;
}

public enum VFMinutiaType: int
{
        Unknown = 0,
        End = 1,
        Bifurcation = 2
}

public struct Minutiae
{
        public int x;
        public int y;
        public VFMinutiaType t;
        public byte d;
        public byte c;
        public byte g;
}

// Features compression
public byte[] FeatSet(int g, int mCount, Minutiae[] m, int spCount,
                SingularPoint[] sp, int boWidth, int boHeight, byte[] bo)
// Features decompression
```

```
public int FeatGetG(byte[] features)
public int FeatGetMinutiaCount(byte[] features)
public Minutiae[] FeatGetMinutiae(byte[] features)
public int FeatGetSPCount(byte[] features)
public SingularPoint[] FeatGetSP(byte[] features)
public int FeatGetBOSize(byte[] features, out int width, out int height)
public byte[] FeatGetBO(byte[] features)
```

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). VeriFinger does not extract singular points
- Blocked orientations - obsolete, obtained using `VFFeatGetBO` function (size of blocked orientations obtained using `VFGetBOSize` function). VeriFinger does not extract blocked orientations

G: 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. You can use `VFFeatG` function to get ridge density.

Minutiae: 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: Singular points are locations 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 byte value in range [`VFDIR_0`, `VFDIR_360`). 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 convert them to radians and vice a versa use `VFDirToRad` and `VFRadToDir` functions. 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: Blocked orientations are ridges orientation of every fingerprint image block of `VF_BLOCK_SIZE` * `VF_BLOCK_SIZE` pixels. Can be byte value range [`VFDIR_0`, `VFDIR_180`), `VFDIR_UNKNOWN` or `VFDIR_BACKGROUND`.

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) ...
```

Delphi:

```
// Conversion from VFDIR_XXX to degrees and vice a versa
function VFDirToDeg(Dir: Byte): Integer;
function VFDirToDegF(Dir: Byte): Double;
function VFDegToDir(Deg: Integer): Byte;
// Conversion from VFDIR_XXX to radians and vice a versa
function VFDirToRad(Dir: Byte): Double;
function VFRadToDir(A: Double): Byte;
// Orientation stuff
function VFIsBadArea(Orient: Byte): Boolean;
function VFIsGoodArea(Orient: Byte): Boolean;
function VFTheOrient(Orient: Byte): Byte;
function VFIsUnknown(Orient: Byte): Boolean;
function VFIsOrient(Orient: Byte): Boolean;
```

Java:

```
public class VeriFingerWrapper {
        ...
        // Directions
        public static double VFDirToDeg(double dir) ...
        public static double VFDegToDir(double deg) ...
        ...
```

8.3. ScanMan library

ScanMan library is a fingerprint scanners manager that enables application to use fingerprints scanning. For a list of supported scanners see Modules and supported scanners.

Typically application can enumerate available scanners and then start capturing from any of it. Also application can monitor scanners plugging and unplugging. Library provides a number of functions to implement this behavior.

Library behavior is controlled through parameters.

Before using ScanMan library has to be initialized.

ScanMan usage from Visual Basic 6.0/.Net/MS Access and Java are described separately (later in this chapter).

8.3.1. Modules and supported scanners

In Light version of VeriFinger SDK ScanMan library contains no modules.

In Standart version of VeriFinger SDK ScanMan library contains following modules:

Module	Support for multiple scan-	Supported scanners

	ners	
UareU	x	DigitalPersona U.are.U Pro™, U.are.U 2000™, U.are.U 4000™
Biometrika		BiometriKa FX2000™
Authentec		AuthenTec AF-S2 FingerLoc™, AES4000 EntrePad™
Ethentica		Ethenticator™ MS 3000, Ethenticator™ USB 2500
ST		STMicroelectronics TCRU1C Reader™
CrossMatch		CrossMatch V300™
Identix		Identix DFR 2090™
Atmel		Atmel scanners
FM200		STARTEK FM200™
Tacoma		Tacoma CMOS™
LighTuning		LighTuning scanner

8.3.2. Library functions

ScanMan library contains the following functions grouped by categories:

Initialization	
SMInitialize	Initializes ScanMan library
SMFinalize	Uninitializes ScanMan library

Parameters	
SMGetParameter	Retrieves parameter value
SMSetParameter	Sets parameter value
Scanner enumeration	
SMGetScannerCount	Retrieves connected scanners count
SMGetScannerId	Retrieves identifier of the scanner
Scanner monitoring	
SMSetMonitor	Sets scanner monitoring
SMRemoveMonitor	Removes scanners monitoring
Capturing	
SMStartCapturing	Starts capturing from the scanner
SMStopCapturing	Stops capturing from the scanner

Each of these functions returns integer value to indicate result of the execution. If it is less than zero then execution of the function failed and the value indicates error code.

You can use `SMFailed` and `SMSucceeded` functions to determine whether the execution of the function failed or succeeded:

C:

```
#define SMFailed(result) ...
#define SMSucceeded(result) ...
```

Delphi:

```
function SMSucceeded(Res: Integer): Boolean;
function SMFailed(Res: Integer): Boolean;
```

8.3.3. Error codes

The following error codes are defined:

General		
`SME_OK`	0	OK, no error
`SME_FAILED`	-1	Failed
`SME_OUT_OF_MEMORY`	-2	Out of memory
`SME_NOT_INITIALIZED`	-3	VeriFinger library is not initialized
`SME_ARGUMENT_NULL`	-4	One of the required function arguments is null
`SME_INVALID_ARGUMENT`	-5	One of the function arguments has an invalid value
Parameters		
`SME_INVALID_PARAMETER`	-10	Parameter identifier is invalid (unknown)
`SME_PARAMETER_READ_ONLY`	-11	Parameter is read only
Capturing		
`SME_NOT_CAPTURING`	-100	Capturing not started for the scanner
`SME_ALREADY_CAPTURING`	-101	Capturing is already started for the scanner

You can use `SMErrorToString` and `SMResultToString` functions to get string that describes error and result. `SMCheckResult` function throws exception in case of the function result indicates failure. These functions are not part of ScanMan library. For C they are implemented in `ScanMan.h` and `ScanMan.cpp` files. For Delphi - in `ScanMan.pas` module.

C:

```
string SMErrorToString(INT error);
string SMResultToString(INT result);
void SMCheckResult(INT result);
```

Delphi:

```
function SMErrorToString(Err: Integer): string;
function SMResultToString(Res: Integer): string;
procedure SMCheckResult(Res: Integer);
```

8.3.4. Initialization

ScanMan library requires initialization to be performed before any function call and uninitialization to be performed after all function calls. This is performed using SMInitialize and SMFinalize functions.

Each successful call to SMInitialize should have a corresponding call to SMFinalize. So you can call SMInitialize more than one time, but you have to call SMFinalize equal number of times.

Example

C:

```
// Main application function
{
        // Application initialization code
        SMInitialize();
        // Other application code
        SMFinalize();
        // Application uninitialization code
}
```

Delphi:

```
// In project source
begin
        // Application initialization code
        SMInitialize;
        // Other application code
        SMFinalize;
        // Application uninitialization code
end.
```

8.3.4.1. SMInitialize function

Initializes ScanMan library.

C:

```
INT SCANMAN_API SMInitialize();
```

Delphi:

```
function SMInitialize: Integer; stdcall;
```

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.In case of error returns `SME_FAILED`.

8.3.4.2. SMFinalize function

Uninitializes ScanMan library if call to the function corresponds to first call to SMInitialize function.

C:

```
INT SCANMAN_API SMFinalize();
```

Delphi:

```
function SMFinalize: Integer; stdcall;
```

Return values: Return value means number of times function should be more called (number of SMInitialize calls without SMFinalize calls).If ScanMan library was not initialized returns `SME_NOT_INITIALIZED`.

8.3.5. Parameters

ScanMan library behavior is controlled through parameters. Parameters are retrieved and set by SMGetParameter and SMSetParameter functions. Some parameters are read only (informational). If you will try to set a read only parameter SMSetParameter function will return `SME_PARAMETER_READ_ONLY`. If you will pass an invalid parameter identifier to one of these functions it will return `SME_INVALID_PARAMETER`.

Parameters can be of the following types:

Referenced as	Size (bytes)	`SM_TYPE_XXX` constant	C equivalent	Delphi equivalent

Void		`SM_TYPE_VOID0`		
Byte	1	`SM_TYPE_BYTE1`	BYTE	Byte
Signed byte	1	`SM_TYPE_SBYTE2`	SBYTE	ShortInt
Word	2	`SM_TYPE_WORD3`	WORD	Word
Short integer	2	`SM_TYPE_SHORT4`	SHORT	SmallInt
Double word	4	`SM_TYPE_DWORD5`	DWORD	LongWord
Integer	4	`SM_TYPE_INT6`	INT	Integer
Boolean	4	`SM_TYPE_BOOL10`	BOOL	Boolean
Char	1	`SM_TYPE_CHAR20`	CHAR	AnsiChar
String	4	`SM_TYPE_STRING100`	CHAR*	PAnsiChar (AnsiString)

To determine parameter type call SMGetParameter function with parameter identifier `SMP_TYPE` and value - needed parameter identifier. Also you may use SMGetParameterType function. Result of the function will be one of `SM_TYPE_XXX` constants.

When retrieving a parameter value pass pointer to variable of parameter type as value for SMGetParameter 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 SMSetParameter function.

In C and Delphi there are functions SMGetXxxParameter and SMSetXxxParameter that work with particular parameter type.

For general parameters there are special functions SMGetXxx defined.

The following parameter identifiers are defined (grouped by categories):

Identifier	Value	Read only	Type	Description
General				
SMP_TYPE	0	x		See parameters types earlier
SMP_NAME	10	x	String	Name of the ScanMan library
SMP_VERSION_HIGH	11	x	Double word	Major version of ScanMan library
SMP_VERSION_LOW	12	x	Double word	Minor version of ScanMan library
SMP_COPYRIGHT	13	x	String	Copyright of ScanMan library

8.3.5.1. SMGetParameter function

Retrieves specified parameter value.

C:

```
INT SCANMAN_API SMGetParameter(INT reserved, INT parameter, VOID * value);
```

Delphi:

```
function SMGetParameter
(
        Reserved: Integer;
        Parameter: Integer;
        Value: Pointer
): Integer; stdcall;
```

Parameters:

	reserved, Reserved	Reserved, must be zero

	parameter, Parameter	Parameter identifier to retrieve
[out]	value, Value	Pointer to variable that will receive parameter value

Return values: If ScanMan library is not initialized returns `SME_NOT_INITIALIZED`.If reserved value is not zero returns `SME_INVALID_ARGUMENT`.If parameter is invalid (unknown) returns `SME_INVALID_PARAMETER`.If value is null returns `SME_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;

        // Get ScanMan library name
        l = SMGetParameter(0, SMP_NAME, NULL);
        name = (CHAR*)malloc((l + 1) * sizeof(CHAR));
        SMGetParameter(0, SMP_NAME, name);
        printf(name);
        free(name);

        // Get ScanMan library major version
        SMGetParameter(0, SMP_VERSION_HIGH, version);
        printf("Version: %u.%u", version, HIWORD(version),
        LOWORD(version));
        // To be supplied
        // ...
}
```

Delphi:

```
// Some application function
var
        Name: AnsiString;
        L: Integer;
        Version: LongWord;
begin
        // Get ScanMan library name
        L := SMGetParameter(0, SMP_NAME, nil);
        SetLength(Name, L + 1);
        SMGetParameter(SMP_NAME, PAnsiString(Name), nil);
        ShowMessage(Name);
        // Get ScanMan library major version
        SMGetParameter(0, SMP_VERSION_HIGH, Version);
        ShowMessage(Format('Version: %u.%u’, [LoWord(Version),
        HiWord(Version)]));
        // To be supplied
        // ...
end;
```

8.3.5.2. SMSetParameter function

Sets specified parameter value.

C:

```
INT SCANMAN_API SMSetParameter(INT reserved, INT parameter, DWORD value);
```

Delphi:

```
function SMSetParameter
(
        Reserved: Integer;
        Parameter: Integer;
        Value: LongWord
): Integer; stdcall;
```

Parameters:

	reserved, Reserved	Reserved, must be zero
	parameter, Parameter	Parameter identifier to set
	value, Value	Parameter value to set

Return values: If ScanMan library is not initialized returns `SME_NOT_INITIALIZED`.If reserved value is not zero returns `SME_INVALID_ARGUMENT`.If parameter is invalid (unknown) returns `SME_INVALID_PARAMETER`.Otherwise returns `VFE_OK`.

Example:

C:

```
// Some application function
{
        // To be supplied
        // ...
}
```

Delphi:

```
// Some application function
begin
        // To be supplied
        // ...
end;
```

8.3.5.3. Additional functions

The following functions are not part of ScanMan library. For C they are implemented in `ScanManX.h` and `ScanManX.cpp` files. For Delphi - in `ScanMan.pas` module.

`SMGetXxxParameter` functions retrieve parameters values of some type.

`SMSetXxxParameter` functions set parameters values of some type.

`SMGetXxx` functions retrieve general parameters values (`SMP_TYPE`, `SMP_NAME`, `SMP_VERSION_HIGH`, `SMP_VERSION_LOW`, `SMP_COPYRIGHT`).

C:

```
// Get
DWORD SMGetDWordParameter(INT reserved, INT parameter);
string SMGetStringParameter(INT reserved, INT parameter);
// Get general
INT SMGetParameterType(INT reserved, INT parameter);
string SMGetName(INT reserved = 0);
DWORD SMGetVersionHigh(INT reserved = 0);
DWORD SMGetVersionLow(INT reserved = 0);
string SMGetCopyright(INT reserved = 0);
```

Delphi:

```
// Get
function SMGetDWordParameter(Reserved: Integer;
                        Parameter: Integer): LongWord;
function SMGetStringParameter(Reserved: Integer;
                        Parameter: Integer): string;
// Get general
function SMGetParameterType(Reserved: Integer; Parameter: Integer): Integer;
function SMGetName(Reserved: Integer = 0): string;
function SMGetVersionHigh(Reserved: Integer = 0): LongWord;
function SMGetVersionLow(Reserved: Integer = 0): LongWord;
function SMGetCopyright(Reserved: Integer = 0): string;
```

Parameters:

	reserved, Reserved	Reserved, must be zero

	parameter, Parameter	Parameter identifier to retrieve or set
	value, Value	Parameter value to set

Return values: `SMGetXxxParameter` functions return specified parameter value.`SMGetXxx` functions return corresponding general parameter value.

Exceptions: All functions raise exception in case of error.

Example:

C:

```
// Some application function
{
        string name;
        DWORD version;
        // Get ScanMan library name
        name = SMGetStringParameter(0, SMP_NAME);
        // or
        name = SMGetName();
        printf(name);
        // Get ScanMan library major version
        version = SMGetDWordParameter(0, SMP_VERSION_HIGH);
        // or
        version = SMGetVersionHigh();
        printf("Version: %u.%u", version, HIWORD(version),
        LOWORD(version));
        // To be supplied
        // ...
}
```

Delphi:

```
// Some application function
var
        Name: AnsiString;
        L: Integer;
        Version: LongWord;
begin
        // Get ScanMan library name
        Name := SMGetStringParameter(0, SMP_NAME);
        // or
        Name := SMGetName;
        ShowMessage(Name);
        // Get ScanMan library major version
        Version := SMGetDWordParameter(0, SMP_VERSION_HIGH);
        // or
        Version := SMGetVersionHigh;
        ShowMessage(Format('Version: %u.%u’, [LoWord(Version),
        HiWord(Version)]));
        // To be supplied
        // ...
end;
```

8.3.6. Scanner enumeration

Application can determine number of connected scanners using SMGetScannerCount function and identifier of each connected scanner using SMGetScannerId function. Also you may use SMGetScannerIds function to get list of connected scanners identifiers. Scanner identifier can be passed to capturing functions. See also Scanner identifiers.

8.3.6.1. SMGetScannerCount function

Retrieves number of connected scanners.

C:

```
INT SCANMAN_API SMGetScannerCount();
```

Delphi:

```
function SMGetScannerCount: Integer; stdcall;
```

Return values: If ScanMan library is not initialized returns `SME_NOT_INITIALIZED`.In case of error return `VFE_FAILED`.Otherwise returns number of connected scanners.

8.3.6.2. SMGetScannerId function

Retrieves identifier of specified scanner. Second overloaded function is not part of ScanMan library. For C it is implemented in `ScanManX.h` and `ScanManX.cpp` files. For Delphi - in `ScanMan.pas` module.

C:

```
INT SCANMAN_API SMGetScannerId(INT index, CHAR *id, INT len);
string SMGetScannerId(INT index);
```

Delphi:

```
function SMGetScannerId
(
        Index: Integer;
        Id: PChar;
        Len: Integer
): Integer; overload;
```

```
function SMGetScannerId(Index: Integer): string; overload;
```

Parameters:

	index, Index	Zero-based index of connected scanner, must be in range from 0 to number of connected scanners minus one. See also SMGetScannerCount
[out]	id, Id	For first overloaded function - pointer to string that receives scanner identifier. If is null then length (including terminating null character) of scanner identifier is returned
	len, Len	For first overloaded function - maximal length of the string (including terminating null character) pointed by id. Maximal number of characters that will be written to the string is the value

Return values: If ScanMan library is not initialized both functions return `SME_NOT_INITIALIZED`.If scanner index is less than zero or greater than or equal to connected scanners count returns `SME_FAILED`.First overloaded function returns either `SME_FAILED` (in case of error) or number of characters written to the string if id (Id) is not null or required length of the string (including the terminating null character) otherwise.Second overloaded function returns string containing the scanner identifier.

Exceptions: Second overloaded function raise exception in case of error.

8.3.6.3. SMGetScannerIds function

Retrieves list of connected scanners identifiers. In case of error list may be empty or may have no identifiers that were failed to retrieve. This function is not part of ScanMan library. For C it is implemented in `ScanManX.h` and `ScanManX.cpp` files. For Delphi - in `ScanMan.pas` module.

C:

```
void SMGetScannerIds(strings &ids);
```

Delphi:

```
procedure SMGetScannerIds(Ids: TStrings);
```

Parameters:

	ids, Ids	Reference to list of strings store identifiers of connected scanners to

8.3.7. Scanner monitoring

Application can use scanner monitoring to receive scanner plug/unplug events or scanner error notifications. To implement this behavior use SMSetMonitor function and pass SMMonitorProc callback that will receive scanner monitoring events. When you want to stop monitoring call SMRemoveMonitor function.

The following events are defined:

`SM_EVENT_PLUGGED`	1	Scanner is plugged
`SM_EVENT_UNPLUGGED`	2	Scanner is unplugged
`SM_EVENT_ERROR`	3	Scanner error

8.3.7.1. SMSetMonitor function

Sets monitoring function

C:

```
INT SCANMAN_API SMSetMonitor(SM_MONITOR_PROC * monitor_proc, VOID * param);
```

Delphi:

```
function SMSetMonitor(MonitorProc: TMonitorProc; Param: Pointer); stdcall;
```

Parameters:

	monitor_proc, MonitorProc	Callback function that will receive scanner monitoring events. See also SMMonitorProc
	param, Param	Parameter passed to SMMonitorProc

Return values: If ScanMan library is not initialized returns `SME_NOT_INITIALIZED`.If monitor_proc (MonitorProc) is null returns `SME_ARGUMENT_NULL`.In case of error returns `SME_FAILED`.Otherwise returns `SME_OK`.

8.3.7.2. SMRemoveMonitor function

Removes monitoring function set by SMSetMonitor. So SMMonitorProc will not be called.

C:

```
INT SCANMAN_API SMRemoveMonitor();
```

Delphi:

```
function SMRemoveMonitor: Integer; stdcall;
```

Return values: If ScanMan library is not initialized returns `SME_NOT_INITIALIZED`.Otherwise returns `SME_OK`.

8.3.7.3. SMMonitorProc callback

SMMonitorProc is a placeholder for application supplied name of callback function that it passes to SMSetMonitor function. This callback is called every time scanner event occurs.

C:

```
typedef VOID SM_CALLBACK SM_MONITOR_PROC
(
        const CHAR * scanner_id,
        DWORD event,
        VOID * param
);
```

Delphi:

```
type
        TSMMonitorProc = procedure
        (
                const ScannerId: PAnsiChar;
                Event: LongWord;
                Param: Pointer
        ); stdcall;
```

Parameters:

	scanner_id, ScannerId	Identifier of the scanner that raised the event
	event, Event	Scanner monitoring event. See earlier

	param, Param	Parameter passed to SMSetMonitor function

8.3.8. Capturing

Application can start capturing for a scanner to receive a fingerprint image (through SMImageProc callback) when user places a finger onto the scanner. Also application can receive information about capturing state (finger placed onto the scanner, finger removed from the scanner, etc.) through SMStateProc callback. To implement this behavior pass these callbacks to SMStartCapturing function (any can be null if you do not need one). When you want to stop capturing (do not receive more images and information about capturing state) call SMStopCapturing function.

The following capturing states are defined:

`SM_STATE_FINGER_DOWN`	1	Finger is placed onto the scanner
`SM_STATE_FINGER_UP`	2	Finger is removed from the scanner

8.3.8.1. SMStartCapturing function

Starts capturing for the specified scanner. Second overloaded function is not part of ScanMan library. For C it is implemented in `ScanManX.h` and `ScanManX.cpp` files. For Delphi - in `ScanMan.pas` module. See also Scanner enumeration.

C:

```
INT SCANMAN_API SMStartCapturing
(
        const CHAR * scanner_id,
        SM_IMAGE_PROC * image_proc,
        SM_STATE_PROC * state_proc,
        VOID * param
);

INT SMStartCapturing
(
        const string &scanner_id,
        SM_IMAGE_PROC * image_proc,
        SM_STATE_PROC * state_proc,
        VOID * param
);
```

Delphi:

```
function SMStartCapturing
(
        ScannerId: PAnsiChar;
        ImageProc: TSMImageProc;
        StateProc: TSMStateProc;
        Param: Pointer
): Integer; stdcall; overload;

function SMStartCapturing
(
        ScannerId: string;
        ImageProc: TSMImageProc;
        StateProc: TSMStateProc;
        Param: Pointer
): Integer; stdcall; overload;
```

Parameters:

	scanner_id, ScannerId	Identifier of the scanner to start capturing for. Have to be a one obtained by scanner enumeration functions
	image_proc, ImageProc	Callback function that will receive images from scanner. See also SMImageProc
	state_proc, StateProc	Callback function that will receive information about capturing states. See earlier
	param, Param	Parameter passed to SMImageProc and SMStateProc

Return values: If ScanMan library is not initialized returns `SME_NOT_INITIALIZED`.If scanner_id (ScannerId) is null returns `SME_ARGUMENT_NULL`.If capturing is already started for the specified scanner returns `SME_ALREADY_CAPTURING`.In case of error returns `SME_FAILED`.Otherwise returns `SME_OK`.

8.3.8.2. SMStopCapturing function

Stops capturing started with SMStartCapturing for specified scanner (or for all scanners). Second and third overloaded functions are not part of ScanMan library. For C they are implemented in `ScanManX.h` and `ScanManX.cpp` files. For Delphi - in `ScanMan.pas` module. See also Scanner enumeration.

C:

```
INT SCANMAN_API SMStopCapturing(const CHAR * scanner_id);
INT SMStopCapturing(const string &scanner_id);
INT SMStopCapturing();
```

Delphi:

```
function SMStopCapturing(ScannerId: PAnsiChar): Integer; stdcall; overload;
function SMStopCapturing(ScannerId: string): Integer; overload;
function SMStopCapturing: Integer; overload;
```

Parameters:

	scanner_id, ScannerId	For first and second overloaded functions - identifier of the scanner to stop capturing for. Have to be a one obtained by scanner enumeration functions. For first function can be `null`: stops capturing for all scanners (same for third function)

Return values: If ScanMan library is not initialized returns `SME_NOT_INITIALIZED`.If capturing is not started for the specified scanner then returns `SME_NOT_CAPTURING`.Otherwise returns `SME_OK`.

8.3.8.3. SMImageProc callback

SMImageProc is a placeholder for application supplied name of callback function that it passes to SMStartCapturing function. This callback is called every time fingerprint image is scanned.

C:

```
typedef VOID SM_CALLBACK SM_IMAGE_PROC
(
        const CHAR * scanner_id,
        INT width,
        INT height,
        const BYTE * image,
        INT resolution,
        VOID * param
);
```

Delphi:

```
type
        TSMImageProc = procedure
        (
                const ScannerId: PAnsiChar;
                Width, Height: Integer;
                const Image: PByte;
                Resolution: Integer;
                Param: Pointer
        ); stdcall;
```

Parameters:

	scanner_id, ScannerId	Identifier of the scanner fingerprint image is scanned from
	width, Width	Width of the scanned image
	height, Height	Height of the scanned image
	image, Image	Fingerprint image
	resolution, Resolution	Resolution of the scanned image in dots per inch (dpi)
	param, Param	Parameter passed to SMStartCapturing function

8.3.8.4. SMStateProc callback

SMStateProc is a placeholder for application supplied name of callback function that it passes to SMStartCapturing function. This callback is called every time capturing state changes.

C:

```
typedef VOID SM_CALLBACK SM_STATE_PROC
(
        const CHAR * scanner_id,
        DWORD state,
        VOID * param
);
```

Delphi:

```
type
TSMStateProc = procedure
(
        const ScannerId: PAnsiChar;
        State: LongWord;
        Param: Pointer
); stdcall;
```

Parameters:

	scanner_id, ScannerId	Identifier of the scanner fingerprint image is scanned

		from
	state, State	Capturing state. See earlier
	param, Param	Parameter passed to SMStartCapturing function

8.3.9. Scanner identifiers

Scanner identifier (id) is a string that describes the scanner. It consists of module name (for example `"UareU"`) and if module supports more than one scanner backslash character (`"\"`) and scanner id (for example `"{xxxx-xxxx-xxxx-xxxx}"` for UareU module) follows. For more information see Modules and supported scanners.

Scanner identifier can be passed to capturing functions and is received in monitor and capturing callbacks.

8.3.10. ScanMan Visual Basic, C# support

Visual Basic 6.0/.Net and Microsoft Access VBA are not designed to work with pointers therefore COM wrapper for ScanMan library was created.

COM wrapper implementation is implemented in `sslCom.dll`. This library has to be registered in system before using. Run `sslCOMreg.bat` file from `\Bin` directory to perform registration. Project must have reference to COM wrapper for start using ScanMan library.

C# demo aplication uses ScanMan COM wrapper.

COM wrapper library exports two classes:

1. `SSLScanner` - scanner class; one class instance represents one physical device;
2. `SSLScannerMan` - scanners manager (enumerates, creates and monitor scanners).

`SSLScannerMan` has following interface (declarations of methods, events and properties are written using Visual Basic 6.0 syntax):

Methods	
Method	Description
`Initialize`	`[Visula Basic]`

	`Sub Initialize()`
	`[C#]`
	`public void Initialize()`
	Initializes ScanMan library. It must be called before starting using ScanMan, typically at the beginning of application.

`Finalize`	`[Visula Basic]`
	`Sub Finalize()`
	`[C#]`
	`public void Finalize()`
	Finalizes ScanMan library. It must be called when application stops using ScanMan library, typically at the end of application.

`EnumDeviceIDs`	`[Visula Basic]`
	`Sub EnumDeviceIDs(IDs)`
	`[C#]`
	`public void EnumDeviceIDs(object IDs)`
	Enumerates all connected and recognized devices. Returns array of strings that contains device identificators.

`CreateScanner`	`[Visula Basic]`
	`Sub CreateScanner(ID As String, Scanner)`
	`[C#]`
	`public void CreateScanner(string ID,object Scanner)`
	Creates scanner object for specified identificator.

`GetErrorDe-`	

`scription`

```
[Visula Basic]
Sub GetErrorDescription(hr As Long, pDescr As String)
```

```
[C#]
public void GetErrorDescription(int hr ,string Descr)
```

Returns description for specified error code.

Properties

Property	Description

DeviceCount

```
[Visula Basic]
Property DeviceCount As Long
```

```
[C#]
public int DeviceCount{get;}
```

Contains connected and recognized device count.

Events

Event	Description

DeviceStatus

```
[Visula Basic]
Event DeviceStatus(Status As Long, ID As String)
```

```
[C#]
public event
_ISSLScannerManEvents_DeviceStatusEventHandler
DeviceStatus;

public delegate
        _ISSLScannerManEvents_DeviceStatusEventHandler
(
        int Status,
        string ID
);
```

	Event is raised when device status is changed (new scanner connected or existing disconnected, etc.). **Posible status values:**

```
1 - New device plugged (FPS_DEVICE_STATUS_PLUGGED)
2 - Device unplugged (FPS_DEVICE_STATUS_UNPLUGGED)
3 - Device status unknown error
                (FPS_DEVICE_STATUS_UNKNOWN_ERROR)
4 - Device status error (FPS_DEVICE_STATUS_ERROR)
```

	ID - contains device id.

`SSLScanner` can be created only with `CreateScanner` method. It has following interface:

Methods	
Method	Description

`StartCapturing`	`[Visula Basic]`
	`void StartCapturing()`
	`[C#]`
	`public void StartCapturing()`
	Starts capturing fingerprint image from associated device.

`StopCapturing`	`[Visula Basic]`
	`void StopCapturing()`
	`[C#]`
	`public void StopCapturing()`
	Stops capturing fingerprint image.

`CaptureOneIm-`	

`age`	`[Visula Basic]`
	`Sub CaptureOneImage(TimeOut As Long)`
	`[C#]`
	`public void CaptureOneImage(int TimeOut)`
	Starts capturing. Capturing will be automatically stopped after finger will be scanned or specified time will end (0 indicates: 'wait infinite time').

`SaveImage`	`[Visula Basic]`
	`Sub SaveImage(File As String)`
	`[C#]`
	`public void SaveImage(string File)`
	Saves scanned image to specified file (BMP).

Properties	
Property	Description

DeviceID	`[Visula Basic]`
	`Property DeviceID As String`
	`[C#]`
	`public string DeviceID{get;}`
	Contains associated device identifier.

Events	
Event	Description
Error	`[Visula Basic]`

```
Event Error(Error As Long, Description As String)
```

```
[C#]
public event
_ISSLScannerEvents_ErrorEventHandler
Error;

public delegate _ISSLScannerEvents_ErrorEventHandler
(
        int Error,
        string Description
);
```

Event is raised when error occurs.

Image

```
[Visula Basic]
Event Image(Width As Long, Height As Long, Image)
```

```
[C#]
public event
_ISSLScannerEvents_ImageEventHandler
Image;

public delegate _ISSLScannerEvents_ImageEventHandler
(
        int Width,
        int Height,
        object Image
);
```

Passed scanned image to application.

Status

```
[Visula Basic]
Event Status(Status As Long)
```

```
[C#]
public event
_ISSLScannerEvents_StatusEventHandler
Status;

public delegate _ISSLScannerEvents_StatusEventHandler
(
        int Status
```

<table>
<tr><td></td><td>

```
);
```

Event is raised when scanner status is changed.

Possible status values:

```
1 - Scanner is waiting for image
                (FPS_STATUS_WAITING_FOR_IMAGE)
2 - Scanner read image (FPS_STATUS_IMAGE_READY)
3 - Finger putted on scanner
                (FPS_STATUS_FINGER_TOUCHING)
4 - Finger removed from scanner
                (FPS_STATUS_FINGER_REMOVED)
5 - Connection error (FPS_STATUS_CONNECTION_ERROR)
```

</td></tr>
</table>

Note: all declarations of methods, events and properties (above) are written using Visual Basic 6.0 syntax. In Visual Basic .Net same methods, events and properties are available but will have different signatures. Type "Long" will be changed to "Integer".

Example:

Visual Basic:

```
Dim WithEvents Scanner As SSLScanner
Dim ScannerManager As SSLScannerMan
...
' main form load sub
Set ScannerManager = New SSLScannerMan
ScannerManager.Initialize
Set Scanner = Nothing
...
' main form unload sub
Set Scanner = Nothing
ScannerManager.Finalize
...
' in some function
Dim ids As Variant
ScannerManager.EnumDeviceIDs ids
If UBound(ids) < LBound(ids) then Exit Sub
Dim vscanner As Variant
ScannerManager.CreateScanner ids(0), vscanner
Set scanner = vscanner
Scanner.StartCapturing
...
```

Visual Basic .Net:

```
Imports SSLCOMLib
' in main form or some module
```

```
Public WithEvents Scanner As SSLScanner = Nothing
Public ScannerManager As SSLScannerMan = Nothing
...
' main form load sub
ScannerManager = New SSLScannerMan()
ScannerManager.Initialize()
Scanner = Nothing
...
' main form closed sub
Scanner = Nothing
ScannerManager.Finalize()
...
' in some function
Dim ids As Object
ScannerManager.EnumDeviceIDs(ids)
If UBound(ids) < LBound(ids) then Exit Sub
Dim vscanner As Object
ScannerManager.CreateScanner(ids(0), vscanner)
Set scanner = vscanner
Scanner.StartCapturing
...
```

C#:

```
//in the project add COM component reference to sslCOM.dll
SSLScannerMan scannerMan = new SSLScannerManClass();
scannerMan.Initialize();

object obj = null;
scannerMan.EnumDeviceIDs(ref obj);
object[] objArr = (object[])obj;
//in some function
public SSLScanner scan;
scannerMan.CreateScanner( objArr[1].ToString(), ref scan);
SSLScanner s = (SSLScanner)scan;
s.StartCapturing();
...
//after work in Dispose method
//stop capturing
scannerMan.Finalize();
```

8.3.11. ScanMan Java support

Java applications can access ScanMan library through special wrapper (`SMJavaW.dll`). Library interface declared in ScanMan class:

Category	Methods
Initialization	`public static native void SMInitialize() throws ScanManException, Exception;`

Initializes ScanMan library. Exception is thrown if error occurs.

```
public static native void SMFinalize()
       throws ScanManException, Exception;
```

Uninitializes ScanMan library if call to the function corresponds to first call to SMInitialize function. Exception is thrown if error occurs.

Parameters

```
// Parameter types
public static final int SM_TYPE_VOID = 0;
public static final int SM_TYPE_BYTE = 1;
public static final int SM_TYPE_SBYTE = 2;
public static final int SM_TYPE_WORD = 3;
public static final int SM_TYPE_SHORT = 4;
public static final int SM_TYPE_DWORD = 5;
public static final int SM_TYPE_INT = 6;
public static final int SM_TYPE_BOOL = 10;
public static final int SM_TYPE_CHAR = 20;
public static final int SM_TYPE_STRING = 100;

// Parameters - general
public static final int SMP_NAME = 10;
public static final int SMP_VERSION_HIGH = 11;
public static final int SMP_VERSION_LOW = 12;
public static final int SMP_COPYRIGHT = 13;
```

```
public static native String getParameterValue
(
        int parameter
) throws ScanManException, Exception;
```

Retrieves specified parameter value (value is converted to String type). Exception is thrown if error occurs.

```
public static native void setParameterValue
(
        int parameter,
        String value
) throws ScanManException, Exception;
```

Set specified parameter value (value must be converted to String type). Exception is thrown if error occurs.

```
public static native int getParameterType
(
        int parameter
) throws ScanManException, Exception;
```

Returns specified parameter type. Exception is thrown if error occurs.

Scanner enumeration

```
public static native int SMGetScannerCount()
      throws ScanManException, Exception;
```

Retrieves number of connected scanners. Exception is thrown if error occurs.

```
public static native String SMGetScannerId
(
        int index
) throws ScanManException, Exception;
```

Retrieves identifier of specified scanner. Possible indexes: from `0` to `SMGetScannerCount()-1`. Exception is thrown if error occurs.

Scanner monitoring

```
public static native void SMSetMonitor
(
        ScannersMonitor monitor
) throws ScanManException, Exception;
```

Sets scanner monitoring. Exception is thrown if error occurs.

Available events:

```
public static final int SM_EVENT_PLUGGED = 1;
// new scanner plugged
public static final int SM_EVENT_UNPLUGGED = 2;
// scanner unplugged
public static final int SM_EVENT_ERROR = 3;
// scanner error
```

Scanner monitoring is performed by calling monitorEvent method of specified object.

	`public static native void SMRemoveMonitor() throws ScanManException, Exception;` Stops scanner monitoring. Exception is thrown if error occurs.

Capturing	`public static native void SMStartCapturing ( String id, ScannerEventListener listener ) throws ScanManException, Exception;`
	Starts capturing for the specified scanner. Scanner is specified by identificator (to obtain it - use `SMGetScannerId` method). Exception is thrown if error occurs. Capturing is performed by calling `imageEvent` and `statusEvent` methods of specified object.
	`public static native void SMStopCapturing ( String id ) throws ScanManException, Exception;`
	Stops capturing for the specified scanner. Scanner is specified by identificator (to obtain it - use SMGetScannerId method). Exception is thrown if error occurs.

If error occurs exception is thrown, you can retrieve error code using getErrorCode method of ScanManException class.

Possible error codes are defined in ScanManException class:

```
public class ScanManException extends Exception {
        ...
        // Error codes:
        // General
        public static final int SME_OK = 0;
        public static final int SME_FAILED = -1;
        public static final int SME_OUT_OF_MEMORY = -2;
        public static final int SME_NOT_INITIALIZED = -3;
        public static final int SME_ARGUMENT_NULL = -4;
        public static final int SME_INVALID_ARGUMENT = -5;
        // Parameters
        public static final int SME_INVALID_PARAMETER = -10;
        public static final int SME_PARAMETER_READ_ONLY = -11;
        // Capturing
        public static final int SME_NOT_CAPTURING = -100;
        public static final int SME_ALREADY_CAPTURING = -101;
        ...
        public int getErrorCode()
        ...
```

Appendix A. Support and Contacts

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:

- `<info@neurotechnologija.com>` for licenses and sales.
- `<linux@neurotechnologija.com>` for Linux and Mac OS X.
- `<support@neurotechnologija.com>` for any help on solving the other development problems.

Appendix B. Distribution Content

VeriFinger SDK distribution contains the following folders and files:

`bin/Win*/`

A directory containing binaries for Microsoft Windows operating system. Currently there is only `Win32_x86` directory for Windows OS running on 32-bit x86 CPU.

`register.bat`	License service registration file.
`FPScannerMan.dll`	FPScannerMan library.
`FPScannerManCom.dll`	FPScannerManCom library.
`FPSmm/FPSmmAtmel.dll`	Atmel FingerChip sensor module for FPScannerMan library.
`FPSmm/FPSmmAuthentec.dll`	Authentec AES2501B, AES4000, AF-S2 sensors module for FPScannerMan library.
`FPSmm/FPSmmBiometrika.dll`	Biometrika FX2000, FX3000 scanners module for FPScannerMan library.
`FPSmm/FPSmmCrossMatch.dll`	CrossMatch V300/V300LC scanner module for FPScannerMan library.
`FPSmm/FPSmmCyte.dll`	
`FPSmm/FPSmmDactyScan.dll`	Green Bit DactyScan 26 scanner module for FPScannerMan library.
`FPSmm/FPSmmDigent.dll`	Digent Izzix 1000 scanner module for FPScannerMan library.
`FPSmm/FPSmmEthentica.dll`	Ethentica scanner module for FPScannerMan library.
`FPSmm/FPSmmFM200.dll`	Startek FM200 scanner module for FPScannerMan library.
`FPSmm/FPSmmFujitsu.dll`	Fujitsu MBF200 scanner module for FPScannerMan library.
`FPSmm/FPSmmFutronic.dll`	Futronic FS80 scanner module for FPScannerMan library.
`FPSmm/FPSmmIdentix.dll`	Identix DFR2080, DFR2090, DFR2100 scanner module for FPScannerMan library.
`FPSmm/FPSmmLighTunning.dll`	LighTunning LTT-C500 scanner module for

	FPScannerMan library.
`FPSmm/FPSmmST.dll`	STMicroelectronics TouchChip TCRU1C sensor module for FPScannerMan library.
`FPSmm/FPSmmTacoma.dll`	Tacoma CMOS scanner module for FPScannerMan library.
`FPSmm/FPSmmUareU.dll`	Digital Persona U.are.U 2000/4000 scanner module for FPScannerMan library.
`FPSmm/FPSmmUpek.dll`	Required for `FPSmm/FPSmmAtmel` module.
`FPSmm/FSM26U.dll`	Required for `FPSmm/DactyScan` module.
`FPSmm/ftrScanAPI.dll`	
`FPSmm/fx3.dll`	Required for `FPSmm/FPSmmBiometrika` module.
`FPSmm/fx3scan.dll`	Required for `FPSmm/FPSmmBiometrika` module.
`FPSmm/GetImageC500.dll`	Required for `FPSmm/FPSmmLighTunning` module.
`FPSmm/Itf32_2080U2.dll`	
`FPSmm/IZZIX.dll`	
`FPSmm/SMZ_API.dll`	Required for `FPSmm/FPSmmTacoma` module.
`FPSmm/SmzCmos1.dll`	Required for `FPSmm/FPSmmTacoma` module.
`FPSmm/TCI.dll`	Required for `FPSmm/FPSmmST` module.
`NCore.dll`	NCore library.
`Neurotec.dll`	Neurotec library.
`Neurotec.Biometrics.FPScannerMan.dll`	Neurotec.Biometrics.FPScannerMan library.
`Neurotec.Biometrics.Gui.NFView.dll`	Neurotec.Biometrics.Gui.NFView library.
`Neurotec.Biometrics.NFRecord.dll`	Neurotec.Biometrics.NFRecord library.

`Neurotec.Biometrics.VFExtractor.dll`	Neurotec.Biometrics.VFExtractor library.
`Neurotec.Biometrics.VFMatcher.dll`	Neurotec.Biometrics.VFMatcher library.
`NFRecord.dll`	NFRecord library.
`NImages.dll`	NImages library.
`ScanMan.dll`	ScanMan library.
`VFExtractor.dll`	VFExtractor library.
`VFinger.dll`	Wrapper with old VeriFinger interface uses VeriFinger 5 components.
`VFMatcher.dll`	VFMatcher library.
`VFVBP.dll`	VeriFinger SDK components Visual Basic Parser.
`pg.exe`	Neurotechnologija protection service file.
`VeriFingerDemo.exe`	Visual C# demo application.
`VeriFingerDemo.exe`	Visual C# demo application with Main Menu.
`VFDemo.bas.exe`	Microsoft Visual Basic 6.0 demo application.
`VFDemo.cpp.exe`	Microsoft Visual C++ demo application.
`VFDemo.MFC.exe`	Microsoft Visual C++ (MFC) demo application.
`VFDemo.mdb`	Database for use with sample applications.

bin/linux_*/

A directory containing binaries for Linux-based operating systems. Currently there is only `linux_x86` directory for Linux OS running on 32-bit x86 CPU.

`pgd`	Neurotechnologija protection service file.

bin/MacOSX_ppc/

`pgd`	Neurotechnologija protection service file.

`bin/MacOSX_x86/`

`pgd`	Neurotechnologija protection service file.

`documentation/`

A directory containing documentation.

`license.html`	License file.
`VeriFinger SDK.chm`	This documentation in CHM format.
`VeriFinger SDK.pdf`	This documentation in PDF format.

`include/`

A directory containing header files. It has subdirectories - `Windows`, `linux` and `MacOSX` which contain almost the same files but with appropriate line-ends for Windows, Mac OS X and Linux operating system accordingly.

`Bmp.h`	Header file for Bmp module.
`NCore.h`	Header file for NCore module.
`NErrors.h`	Header file for NErrors module.
`NFRecord.h`	Header file for NFRecord module.
`NFRecordV1.h`	Additional header file for NFRecord module.
`NGrayscaleImage.h`	Header file for NGrayscaleImage module.
`NImage.h`	Required for `NImagePro.h`.
`NImageFormat.h`	Required for `NImageFormatPro.h`.
`NImages.h`	Required for `NImagesPro.h`.
`NMonochromeImage.h`	Header file for NMonochromeImage module.

`NParameters.h`	Header file for NParameters module.
`NPixelFormat.h`	Header file for NPixelFormat module.
`NRgbImage.h`	Header file for NRgbImage module.
`NTypes.h`	Header file for NTypes module.
`Tiff.h`	Header file for Tiff module.
`VFExtractor.h`	Header file for VFExtractor module.
`VFExtractorParams.h`	Required for `VFExtractor.h` header.
`VFExtractorTypes.h`	Required for `VFExtractor.h` header.
`VFMatcher.h`	Header file for VFMatcher module.
`VFMatcherParams.h`	Required for `VFMatcher.h` header.
`VfmMatchDetails.h`	Required for `VFMatcher.h` header.

include/Windows/

A directory containing other header files than include/MacOSX/ and include/linux/.

`VFinger.bas`	VeriFinger SDK components module for Visual Basic 6.0
`FPScanner.h`	FPScannerMan library module for Delphi.
`FPScannerMan.h`	FPScannerMan library header file for C/C++ compilers.
`ScanMan.h`	ScanMan library header file for C/C++ compilers.
`VFinger.h`	VeriFinger SDK components header for C/C++ compilers.
`ScanMan.pas`	ScanMan library module for Delphi.
`VFinger.pas`	VeriFinger SDK components module for Delphi.

install/

A directory containing files to install.

`install/Fingerprint Scanners/`	A directory containing drivers for fingerprint scanners.

`lib/Win*/`

A directory contains library and/or DLL import library files for Microsoft Windows operating system. Currently there is only `Win32_x86` directory for Windows OS running on 32-bit x86 CPU.

`FPScannerMan.dll.lib`	FPScannerMan library file.
`NCore.dll.lib`	Import library for NCore library.
`NFRecord.dll.lib`	Import library for NFRecord library.
`NImages.dll.lib`	Import library for NImages library.
`ScanMan.dll.lib`	ScanMan.dll library file.
`VFExtractor.dll.lib`	Import library for VFExtractor library.
`VFinger.dll.lib`	Wrapper with old VeriFinger interface uses VeriFinger 5 components.
`VFMatcher.dll.lib`	Import library for VFMatcher library.

`lib/linux_*/`

A directory containing library files for Linux-based operating systems. Currently there is only `linux_x86` directory for Linux OS running on 32-bit x86 CPU.

`libNCore.so`	NCore library.
`libNFRecord.so`	NFRecord library.
`libNImages.so`	NImages library.
`libVFExtractor.so`	VFExtractor library.
`libVFMatcher.so`	VFMatcher library.

`lib/MacOSX_ppc/`

A directory containing library files for Mac OS X operating systems. Currently there is only `linux_x86` directory for Linux OS running on 32-bit x86 CPU.

`libNCore.dylib`	NCore library.
`libNFRecord.dylib`	NFRecord library.
`libNImages.dylib`	NImages library.
`libVFExtractor.dylib`	VFExtractor library.
`libVFMatcher.dylib`	VFMatcher library.

lib/MacOSX_x86/

A directory containing library files for Mac OS X operating systems.

`libNCore.dylib`	NCore library.
`libNFRecord.dylib`	NFRecord library.
`libNImages.dylib`	NImages library.
`libVFExtractor.dylib`	VFExtractor library.
`libVFMatcher.dylib`	VFMatcher library.

samples/dotNET/

.NET samples.

`/VeriFingerDemo/`	A directory containing source of VeriFinger SDK samle for .NET.
`/VeriFingerDemoMainMenu/`	A directory containing source of VeriFinger SDK samle for .NET with Main Menu.

samples/linux

Linux samples and drivers directory.

`linux/console/`	A directory containing source of VeriFinger SDK not graphic(console) samle.
`linux/images/`	A directory containing images for not graphic(console) sample.
`linux/gtk2/`	A directory containing source of VeriFinger

	SDK the basic demo program for Linux.
`linux/scanners/`	Linux scanners drivers.
`linux/scanners/collector/`	Collects fingerprints images from scanner and saves to current directory.
`linux/scanners/continues_scan/`	Application that continuously scans and saves images.
`linux/scanners/live-demo/`	Application that continuously scans, shows and saves images.
`linux/scanners/simple_scan/`	Application that continuously scans and saves images.

samples/MacOSX_ppc

Mac OS X sample and drivers directory for Power PC.

`/Drivers/`	A directory containing scanners drivers.

samples/MacOSX_x86

Mac OS X sample and drivers directory for running on 32-bit x86 CPU.

`/Drivers/`	A directory containing scanners drivers.

samples/Windows

A directory containing samples and wrappers for Microsoft Windows operating system.

`/VFDemo.bas/`	Source of Visual Basic 6.0 application.
`/VFDemo.cpp/`	Source of Microsoft Visual C++ (later referenced as C) application.
`/VFDemo.java/`	Source of Sun Java 2 application.
`/VFDemo.MFC/`	Source of Microsoft Visual C++ (MFC) application.
`/VFDemo.pas/`	Source of Borland Delphi 6 application.
`/Wrappers/ScanMan Java Wrap-`	ScanMan library wrapper for Java applica-

`per/`	tions.
`/Wrappers/VeriFinger Java Wrapper/`	VeriFinger library wrapper for Java applications.
`/Wrappers/Visual Basic Parser/`	Visual Basic Parser.

Appendix C. Change Log

This appendix lists VeriFinger SDK components changes among versions.

Legend

- FIX - bug was fixed.
- CHN - some changes were made.
- UPD - something has been updated.
- ADD - something has been added.
- REM - something has been removed.

Version 5.0.0.7

- ADD: Identix DFR2100U2 and DFR-2080U2 scanner support added.
- ADD: Green Bit DactyScan 26 scanner support added.

Version 5.0.0.6

- UPD: NCore library to version 2.1.0.1.
- UPD: NImages library to version 2.0.0.4.
- UPD: Updated documentation.

Version 5.0.0.5

- UPD: Neurotec.Images library to version 2.0.2.0.
- UPD: VFExtractor library to version 5.0.0.3.
- UPD: VFMatcher library to version 5.0.0.3.
- UPD: Updated documentation.

Version 5.0.0.4

- ADD: Missing FPSmmNitgen library version 1.0.0.0 as a substitution of sslNitgen library in VeriFinger 4.2.
- ADD: Missing FPSmmSecugen library version 1.0.0.0 as a substitution of sslSecugen library in VeriFinger 4.2.
- UPD: Updated documentation.

Version 5.0.0.3

- UPD: Updated documentation.

Version 5.0.0.2

- UPD: FPScannerMan library to version 1.1.0.2.
- UPD: VFExtractor library to version 5.0.0.2.
- UPD: VFMatcher library to version 5.0.0.2.
- UPD: Updated documentation.

Version 5.0.0.1

- UPD: FPScannerMan library to version 1.1.0.1.
- UPD: VFExtractor library to version 5.0.0.1.
- UPD: VFMatcher library to version 5.0.0.1.
- UPD: VeriFinger Demo .NET sample to version 5.0.0.1.
- ADD: Missing VFDemo.bas Windows sample version 5.0.0.0.
- UPD: VFDemo.cpp Windows sample to version 5.0.0.1.
- ADD: Missing VFDemo.MFC Windows sample version 5.0.0.0.
- UPD: VFDemo.pas Windows sample to version 5.0.0.1.
- ADD: Missing Visual Basic Parser version 5.0.0.0.
- UPD: Updated documentation.

Version 5.0.0.0

- ADD: FPScannerMan library version 1.1.0.0 as a substitution of ssl and ScanMan libraries.
- ADD: FPScannerManCom library version 1.0.0.1 as a substitution of sslCom library.
- ADD: FPSmmAtmel library version 1.0.0.2 as a substitution of sslAtmel library.
- ADD: FPSmmAuthentec library version 1.0.0.0 as a substitution of sslAuthentec library.
- ADD: FPSmmBiometrika library version 1.0.0.2 as a substitution of sslBiometrika library.
- ADD: FPSmmCyte library version 1.0.0.0 as a substitution of sslCyte library.
- ADD: FPSmmCrossMatch library version 1.0.0.3 as a substitution of sslCrossMatch library.
- ADD: FPSmmDigent library version 1.0.0.0 as a substitution of sslDigent library.
- ADD: FPSmmEthentica library version 1.0.0.0 as a substitution of sslEthentica library.
- ADD: FPSmmFM200 library version 1.0.0.2 as a substitution of sslFM200 library.
- ADD: FPSmmFujitsu library version 1.0.0.0 as a substitution of sslFujitsu library.
- ADD: FPSmmFutronic library version 1.0.0.0 as a substitution of sslFutronic library.
- ADD: FPSmmIdentix library version 1.0.0.2 as a substitution of sslIdentix library.
- ADD: FPSmmLighTunning library version 1.0.0.2 as a substitution of sslLighTunning library.
- ADD: FPSmmTacoma library version 1.0.0.2 as a substitution of sslTacoma library.
- ADD: FPSmmUareU library version 1.0.1.0 as a substitution of sslUareU library.
- ADD: FPSmmUpek library version 1.0.0.2 as a substitution of sslUpek library.
- ADD: NCore library version 2.1.0.0.
- ADD: NFRecord library version 1.2.0.0.
- ADD: NImages library version 2.0.0.3.
- ADD: Neurotec library version 2.1.0.0.
- ADD: Neurotec.Biometrics.FPScannerMan library version 1.0.0.2.
- ADD: Neurotec.Biometrics.Gui.NFView library version 1.1.0.1.

- ADD: Neurotec.Biometrics.NFRecord library version 1.2.0.0.
- ADD: Neurotec.Biometrics.VFExtractor library version 5.0.0.0.
- ADD: Neurotec.Biometrics.VFMatcher library version 5.0.0.0.
- ADD: Neurotec.Images library version 2.0.1.0.
- UPD: ScanMan library to version 0.9.5.0.
- ADD: VFExtractor library version 5.0.0.0.
- ADD: VFMatcher library version 5.0.0.0.
- UPD: VFinger library to version 5.0.0.0.
- ADD: VeriFinger Demo .NET sample version 1.0.0.0.
- UPD: VFDemo.cpp Windows sample to version 5.0.0.0.
- UPD: VFDemo.pas Windows sample to version 5.0.0.0.

Version 4.2.1.10

- ADD: BiometriKa FX3000 support added.
- UPD: BiometriKa FX2000 support updated.
- ADD: UPEK TCRU2C support added.
- UPD: UPEK TCRU1C support updated.
- ADD: NitGen Fingkey Hamster and HamsterII support added.
- ADD: AuthenTec AES2501 support added.
- UPD: Visual Basic 6 sample updated (minor fixes).
- ADD: Database checking sample added into "Support" folder.

Version 4.2.1.9

- FIX: Minor bugs fixed.
- ADD: Futronic FS80 support added.

Version 4.2.1.8

- ADD: Testech Bio-I scanner support added.
- CHN: "Install" folder structure changed.

Version 4.2.1.7

- CHN: Minor changes in SDK samples.
- ADD: Added DigitalPersona drivers patch.

Version 4.2.1.6

- ADD: Fujitsu MBF200 support added.
- UPD: Minor changes in SDK samples.
- UPD: Updated HASP drivers (version 4.99 included into SDK).

Version 4.2.1.5

- ADD: Digent Izzix FD1000 scanner support added.

Version 4.2.1.4

- UPD: VFinger library to version 4.2.1.2.

Version 4.2.1.3

- ADD: Atmel, Tacoma CMOS, STARTEK FM200, LighTuning scanners support added.

Version 4.2.1.2

- ADD: Visual C++ MFC sample added.

Version 4.2.1.1

- UPD: VFinger library to version 4.2.1.1.

Version 4.2.1.0

- UPD: VFinger library to version 4.2.1.0.
- ADD: SDK description in CHM format added into VeriFinger SDKs.
- UPD: Minor updates in SDK samples.

Version 4.2.0.3

- UPD: VFinger library to version 4.2.0.3.
- FIX: Fixed VB.NET sample application project.

Version 4.2.0.2

- UPD: VFinger library to version 4.2.0.2.
- FIX: Fixed identification thread bug in C++ and Delphi sample applications.

Version 4.2.0.1

- UPD: VFinger library to version 4.2.0.1.

- FIX: Fixed XML parser header file in C++ sample application.
- UPD: Minor formatting updates in documentation.

Version 4.2.0.0

- UPD: VFinger library to version 4.2.0.0.
- UPD: Added separate "What's new" sections for VeriFinger library, VeriFinger SDK and VeriFinger SDK documentation.

Version 4.1.0.0

- UPD: VFinger library to version 4.1.0.0.
- ADD: ScanMan library version 1.0.0.0 as a substitution of UrUReader, ASReader and FX2KReader in VeriFinger SDK 4.0. Now you can work with any scanner it supports via the same interface.

C.1. Components

C.1.1. FPScannerMan Library

Version 1.1.0.2

- UPD: Minor updates.

Version 1.1.0.1

- FIX: Minor fixes.

Version 1.1.0.0

- ADD: C interface. See Section 6.5, “FPScannerMan Library”.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.2

- UPD: Minor updates.

Version 1.0.0.1

- FIX: Internal synchronization logic.

Version 1.0.0.0

Initial release.

C.1.1.1. FPSmmAtmel Library

Version 1.0.0.2

- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.1

- UPD: Minor updates.

Version 1.0.0.0

Initial release.

C.1.1.2. FPSmmAuthentec Library

Version 1.0.0.0

Initial release.

C.1.1.3. FPSmmBiometrika Library

Version 1.0.0.2

- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.1

- UPD: Minor updates.

Version 1.0.0.0

Initial release.

C.1.1.4. FPSmmCyte Library

Version 1.0.0.0

Initial release.

C.1.1.5. FPSmmCrossMatch Library

Version 1.0.0.3

- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.2

- UPD: Minor updates.

Version 1.0.0.1

- FIX: Errors with synchronization logic.

Version 1.0.0.0

Initial release.

C.1.1.6. FPSmmDigent Library

Version 1.0.0.0

Initial release.

C.1.1.7. FPSmmEthentica Library

Version 1.0.0.0

Initial release.

C.1.1.8. FPSmmFM200 Library

Version 1.0.0.2

- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.1

- UPD: Minor updates.

Version 1.0.0.0

Initial release.

C.1.1.9. FPSmmFujitsu Library

Version 1.0.0.0

Initial release.

C.1.1.10. FPSmmFutronic Library

Version 1.0.0.0

Initial release.

C.1.1.11. FPSmmIdentix Library

Version 1.0.0.2

- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.1

- UPD: Minor updates.

Version 1.0.0.0

Initial release.

C.1.1.12. FPSmmLighTunning Library

Version 1.0.0.2

- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.1

- UPD: Minor updates.

Version 1.0.0.0

Initial release.

C.1.1.13. FPSmmNitgen Library

Version 1.0.0.0

Initial release.

C.1.1.14. FPSmmSecugen Library

Version 1.0.0.0

Initial release.

C.1.1.15. FPSmmTacoma Library

Version 1.0.0.2

- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.1

- UPD: Minor updates.

Version 1.0.0.0

Initial release.

C.1.1.16. FPSmmUareU Library

Version 1.0.1.0

- ADD: Latent previous finger removal capability.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.1

- UPD: Minor updates.

Version 1.0.0.0

Initial release.

C.1.1.17. FPSmmUpek Library

Version 1.0.0.2

- CHN: Renamed from FPSmmST to FPSmmUpek.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.1

- UPD: Minor updates.

Version 1.0.0.0

Initial release.

C.1.2. FPScannerManCom Library

Version 1.0.0.1

- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.0.0.0

Initial release.

C.1.3. NCore Library

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.1.4. NFRecord Library

Version 1.2.0.0

- ADD: CbeffProductType get/set capability, which can be used to distinguish between ex-

tractors that generated the NFRecord.
- UPD: Updated for VeriFinger 5.0 algorithm.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 1.1.0.0

- CHN: Some changes in interface.
- ADD: Editing support.

Version 1.0.2.0

- ADD: Packing to version 1.0 functionality.

Version 1.0.1.1

- UPD: Minor updates.

Version 1.0.1.0

- UPD: Made compatible with new functionality of NFExtractor library version 1.0.1.0.
- ADD: Four-neighbours ridge count support.
- CHN: Flags definition.
- FIX: Minor fixes.

Version 1.0.0.2

- FIX: NFSelectMinutiaeNeighbours parameter list.

Version 1.0.0.1

- FIX: NFRecordGetPositionMem, NFRecordGetImpressionTypeMem, NFRecordGetPatternClassMem, NFRecordGetQualityMem, NFRecordGetGMem functions retrieve correct value from packed NFRecord.
- CHN: NFSelectMinutiaeNeighbours takes flags as last parameter.
- ADD: Additional flag.

Version 1.0.0.0

Initial release.

C.1.5. NImages Library

Version 2.0.0.4

- FIX: Fixed some bad-formed BMP files reading.

Version 2.0.0.3

Initial release.

C.1.6. Neurotec Library

Version 2.1.0.0

- REM: LicenseManagerException class.
- CHN: Now uses Microsoft .NET Framework 2.0.

Version 2.0.1.2

- FIX: Minor fixes in parameters framework.

Version 2.0.1.1

- FIX: Minor fixes.
- UPD: Minor updates in structures.

Version 2.0.1.0

- ADD: NParameters class 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: All error codes are mapped to appropriate exceptions.

Version 1.0.0.0

Initial release.

C.1.7. Neurotec.Biometrics.FPScannerMan Library

Version 1.0.0.2

- UPD: Minor updates.

Version 1.0.0.1

- FIX: Shutdown errors.

Version 1.0.0.0

Initial release.

C.1.8. Neurotec.Biometrics.Gui.NFView Library

Version 1.1.0.1

- CHN: Now uses Microsoft .NET Framework 2.0.

Version 1.1.0.0

- ADD: Events enabling minutiae editing.
- ADD: Minutia selection and neighbours displaying capability.
- CHN: Some changes in internal logic.

Version 1.0.0.1

- FIX: Minor fixes.

Version 1.0.0.0

Initial release.

C.1.9. Neurotec.Biometrics.NFRecord Library

Version 1.2.0.0

- ADD: CbeffProductType property, which can be used to distinguish between extractors that generated the NFRecord.
- CHN: Now uses Microsoft .NET Framework 2.0.

Version 1.1.0.0

- ADD: Editing support.

Version 1.0.2.0

- ADD: Packing to version 1.0 functionality.

Version 1.0.1.1

- UPD: Minor updates.

Version 1.0.1.0

- ADD: Four-neighbours ridge count support.

Version 1.0.0.1

- FIX: NFSelectMinutiaeNeighbours parameter list.

Version 1.0.0.0

Initial release.

C.1.10. Neurotec.Biometrics.VFExtractor Library

Version 5.0.0.0

- ADD: VFExtractor.UseQuality and VFExtractor.QualityThreshold properties and VFExtractor.ParameterUseQuality and VFExtractor.ParameterQualityThreshold constants for controlling input image quality control.
- CHN: Now VFExtractor.Extract/VFExtractor.ExtractUnpacked functions return zero-sized (null) template if image quality is low.
- ADD: VFExtractor.Generalize and VFExtractor.GeneralizaUnpacked methods for generalization of N templates.
- ADD: VFExtractor.TemplateSize property and VFExtractor.ParameterTemplateSize constant and VfeTemplateSize enum controling template size/matcher reliability tradeoff.
- ADD: More VFExtractor.Mode... constants enabling optimizations for more fingerprint scanners.
- CHN: Now uses Microsoft .NET Framework 2.0.

Version 4.2.1.0

- ADD: Extraction of unpacked template capability.
- CHN: Parameters interface.
- CHN: Major version number made consistent with VeriFinger major version.

Version 1.0.0.1

- FIX: Minor fixes.

Version 1.0.0.0

Initial release.

C.1.11. Neurotec.Biometrics.VFMatcher Library

Version 5.0.0.0

- REM: VFMatcher.MatchingSpeed property and VFMatcher.ParameterMatchingSpeed and constant and VfmSpeed enum.
- ADD: More VFMatcher.Mode... constants enabling optimizations for more fingerprint scanners.
- CHN: Now uses Microsoft .NET Framework 2.0.

Version 4.2.0.0

Initial release, separated from Neurotec.Biometrics.NMatcher library.

C.1.12. Neurotec.Images Library

Version 2.0.2.0

- ADD: NImages.GetOpenFileFilter, NImages.GetSaveFileFilter, NImages.GetOpenFileFilterString and NImages.GetSaveFileFilterString methods.
- FIX: Reading of read-only files.

Version 2.0.1.0

Initial release.

C.1.13. ScanMan Library

Version 0.9.5.0

- CHN: Is now obsolete. Use the FPScannerMan library instead.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

C.1.14. VFExtractor Library

Version 5.0.0.4

- FIX: Small template size is not small when quality is turned on.

Version 5.0.0.3

- UPD: Minor updates.

Version 5.0.0.2

- UPD: Minor updates.

Version 5.0.0.1

- FIX: Minor fixes.

Version 5.0.0.0

- UPD: More relailable algorithm than in VeriFinger 4.2.
- UPD: Better quality of input image control.
- ADD: VFEP_USE_QUALITY and VFEP_QUALITY_THRESHOLD constants and parameters for controlling input image quality control.
- CHN: Now VfeExtract.../VfeExtractUnpacked... functions return zero-sized template if image quality is low.
- ADD: VFGeneralize and VFGeneralizaUnpacked functions for generalization of N templates.
- ADD: VFEP_TEMPLATE_SIZE constant and parameter and VfeTemplateSize enum controling template size/matcher reliability tradeoff.
- ADD: More VFEP_MODE_... constants enabling optimizations for more fingerprint scanners.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 4.2.1.0

- ADD: Extraction of unpacked template capability.
- UPD: Minor updates.
- CHN: Major version number made consistent with VeriFinger major version.

Version 1.0.0.2

- FIX: Minor fixes.

Version 1.0.0.1

- UPD: U.are.U scanner mode.

Version 1.0.0.0

Initial release. Contains features extraction functionality of VeriFinger 4.2.

C.1.15. VFMatcher Library

Version 5.0.0.3

- UPD: Minor updates.

Version 5.0.0.2

- UPD: Minor updates.

Version 5.0.0.1

- FIX: Minor fixes.

Version 5.0.0.0

- CHN: The algorithm has one speed and is faster than high speed and more reliable than low speed in VeriFinger 4.2.
- REM: VFMP_MATCHING_SPEED constant and parameter and VfmSpeed enum.
- ADD: More VFMP_MODE_... constants enabling optimizations for more fingerprint scanners.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 4.2.1.0

Initial release, separated from NMatcher library. Contains features matching functionality of VeriFinger 4.2.

C.1.16. VFinger Library

Version 5.0.0.0

- CHN: Is now obsolete. Use VFExtractor and VFMatcher libraries instead.
- UPD: Updated to use Microsoft Visual C++ Runtime Library 8.0.

Version 4.2.1.3

- FIX: Minor bugfixes.

Version 4.2.1.2

- FIX: Features extraction memory leak fix.
- UPD: Matching and generalization memory optimizations.
- UPD: Updated U.are.U scanner mode.

Version 4.2.1.1

- FIX: Minor bugfixes.

Version 4.2.1.0

- UPD: Atmel scanner mode.

Version 4.2.0.3

- FIX: Features extraction with low quality image bug fix.

Version 4.2.0.2

- FIX: Features generalization bug fix.
- FIX: Minor bugfixes.

Version 4.2.0.1

- FIX: Minor bugfixes.

Version 4.2.0.0

- UPD: Improved reliability.
- UPD: Better matching performance. Both matching speeds are now about 50% faster than in version 4.1.
- UPD: Reduced template size. Now template occupies 150 - 300 bytes (vs. 200 - 650 in version 4.1).
- ADD: Features compression and decompression functions are now built in the library.
- ADD: Added CrossMatch Verifier 300 scanner mode.
- ADD: VeriFinger can now return skeletonized image.

Version 4.1.0.0

- ADD: Completely new interface.
- CHN: Higher matching speed. Now only two speeds are available - low (0 speed in 4.0) and high (5 speed in 4.0). Both speeds are faster that in version 4.0.
- UPD: Improved recognition reliability. Both speeds are more reliable that in version 4.0.
- ADD: Optimizations for fingerprint scanners. Optimizations for a number of scanners are available.

C.2. Samples

C.2.1. .NET

C.2.1.1. VeriFingerDemo

Version 5.0.0.1

- UPD: Minor updates.
- UPD: Major version to be as in VeriFinger.

Version 1.0.0.0

Initial release.

C.2.2. Windows

C.2.2.1. VFDemo.Access

Version 5.0.0.0

- CHN: Updated to be compatible with VeriFinger 5.0.

C.2.2.2. VFDemo.bas

Version 5.0.0.0

- CHN: Updated to be compatible with VeriFinger 5.0.

C.2.2.3. VFDemo.cpp

Version 5.0.0.1

- REM: Unneeded registration stuff.

Version 5.0.0.0

- CHN: Updated to be compatible with VeriFinger 5.0.

C.2.2.4. VFDemo.MFC

Version 5.0.0.0

- CHN: Updated to be compatible with VeriFinger 5.0.

C.2.2.5. VFDemo_pas

Version 5.0.0.1

- REM: Unneeded registration stuff.

Version 5.0.0.0

- CHN: Updated to be compatible with VeriFinger 5.0.

C.2.2.6. Wrappers

C.2.2.6.1. Visual Basic Parser

Version 5.0.0.0

- CHN: Updated to be compatible with VeriFinger 5.0.

Glossary

DBMS	Database Management System.
SDK	Software Development Kit.
OS	Operating system.
ppc	Power PC.