VeriFinger 4.2 SDK

VeriFinger 4.2 SDK

Table of Contents

Chapter 1. Introduction

VeriFinger SDK consists of VeriFinger library, ScanMan library and sample applications:

- VeriFinger library is a fingerprint recognition engine.
- ScanMan library provides scanning from fingerprint scanners.
- Sample programs demonstrate how it all works together.

VeriFinger library and ScanMan library stores fingerprint image in format described in Fingerprint images.

VeriFinger SDK can be used in a number of compilers under any of these operating systems: Windows 95/98/Me and Windows NT/2000/XP. VeriFinger and ScanMan libraries are provided as Win32 dynamic link libraries (DLL) `VFinger.dll` and `ScanMan.dll` (in `Bin` directory). Also VeriFinger library can be provided as not protected library - `VFinger.NP.lib` file to use with Microsoft Visual C++ 6.0/7.0/7.1 (in `Lib` directory).

Libraries' interface and sample applications are available for the following compilers:

- Microsoft Visual C++ 6.0/7.0/7.1 (later referenced as C). Header files `VFinger.h` and `ScanMan.h` (in `Include` subdirectory) and DLL library file `VFinger.dll.lib` (for .lib version of VeriFinger library - `VFinger.NP.lib` file) and `ScanMan.lib` (in `Lib` directory) have to be included in your application project. If you are using a C/C++ compiler other than Microsoft Visual C++ 6.0/7.0/7.1 or higher you may have to create another `VFinger.lib` and `ScanMan.lib` files. For example for Borland C/C++ compiler you may use `implib.exe` utility from Borland to produce .lib files from .dll files. Also you may have to change `__cplusplus` identifier to one that indicates C++ compilation and `__stdcall` identifier to one that identifies standard calling convention for your compiler (in `VFinger.h` and `ScanMan.h` line `#ifdef __cplusplus` and line `#define VFINGER_API __stdcall` and line `#define SCANMAN_API __stdcall`) and types defined in `VFinger.h` and `ScanMan.h`. Additional functions are implemented in files `VFingerX.h`, `VFingerX.cpp` and `ScanManX.h` and `ScanManX.cpp` (is sample application directory). For other compilers they could have to be changed. Sample application is available in `VFDemo.cpp` and `VFDemo.dll` directories (for not protected library version of VeriFinger library - in `VFDemo.cpp` and `VFDemo.NP.lib` directories).
- Borland Delphi 6 (further referenced as Delphi). Modules `VFinger.pas` and `ScanMan.pas` (in `Include` directory) have to be included in your application project. All additional functions are implemented in these files. Sample application is available in `VFDemo.pas` directory.

- Microsoft Visual Basic 6.0 (further referenced as Visual Basic). Module `VFinger.bas` (in `Include` directory) have to be included to your application project. This module is using VeriFinger parser `VFVB42.dll` (in `Bin` directory). In demo program (`VFinger.bas` directory) you can find `Service.bas` and `DataBase.bas` files, which can be helpful in developing your application(s).

- Microsoft Visual Basic .Net (further referenced as Visual Basic .Net). Module `VFinger.vb` (in `Include` directory) have to be included to your application project. This module is using the same VeriFinger parser as Visual Basic 6.0 does (`VFVB42.dll`). In demo program (`VFinger.Net.vb` directory) you can find `Service.vb` and `DataBase.vb` files, which can be helpful developing your application(s). Visual Basic .Net sample is very similar to Visual Basic 6 sample therefore only differences will be noticed in this documentation.

- Microsoft Access 2000 (further referenced as MS Access). Code and database are located in the same file (database). MS Access uses same VeriFinger parser as Visual Basic samples (`VFVB42.dll`). Sample contains modules that are very similar to Visual Basic 6 modules. VeriFinger functions are also declared in the same way as in Visual Basic 6 sample therefore only differences will be discussed in this documentation.

- Sun Java 2 (further referenced as Java). 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.

SDK directory contains:

`Bin\`		Directory with binaries
	`VFinger.dll`	VeriFinger library
	`ScanMan.dll`	ScanMan library
	`ssl.dll`	ScanMan library core
	`sslCom.dll`	COM interface for ScanMan library
	`sslCOMreg.bat`	Command file to register `sslCom.dll`

	sslFile.dll	ScanMan File module
	sslAuthentec.dll	ScanMan Authentec module (not available in "Light" version of SDK)
	sslUrU.dll	ScanMan DigitalPersona U.are.U module (not available in "Light" version of SDK)
	fx3.dll fx3scan.dll	ScanMan Biometrika module files (not available in "Light" version of SDK)
	sslBiometrika.dll	ScanMan Biometrika module (not available in "Light" version of SDK)
	sslST.dll	ScanMan STMicroelectronics module (not available in "Light" version of SDK)
	TCI.dll	ScanMan STMicroelectronics module (not available in "Light" version of SDK)
	sslEthentica.dll	ScanMan Ethentica module (not available in "Light" version of SDK)
	sslIdentix.dll	ScanMan Identix DFR 2090 module (not available in "Light" version of SDK)
	sslCrossMatch.dll	ScanMan CrossMatch V300 (not available in "Light" version of SDK)
	VFDemo.mdb	Database for use with sample applications
	VFDemo.dll.cpp.exe	Sample application for Microsoft Visual C++ 6.0/7.0/7.1 (for DLL version of VeriFinger SDK only)
	VFDemo.NP.lib.exe	Sample application for Microsoft Visual C++ 6.0/7.0/7.1

		(for not protected library version of VeriFinger SDK only)
	VFDemo.settings	Settings file for Microsoft Visual C++ 6.0/7.0/7.1 sample applications (created on first use)
	VFDemo.pas.exe	Sample application for Borland Delphi 6
	VFDemo.bas.exe	Sample application for Microsoft Visual Basic 6.0
	VFVBP42.dll	VeriFinger 4.2 Visual Basic Parser
	VFDemo.jar	Compiled VeriFinger sample for Sun Java 2
	VFDemoJava.bat	Batch file that runs Java sample
	VFJavaW42.dll	VeriFinger 4.2 wrapper for Java programs
	SMJavaW.dll	ScanMan wrapper for Java programs
	vfinger.db	Java sample fingerprint database
	VFDemo.net.vb.exe	Sample application for Visual Basic .NET
	AxInterop.*.dll Interop.*.dll	Support libraries for Visual Basic .NET sample application
	vfdemo.cfg	Java sample configuration
Include\		Directory with header files
	VFinger.h	VeriFinger library header for C/C++ compilers

	VFinger.pas	VeriFinger library module for Delphi
	ScanMan.h	ScanMan library header file for C/C++ compilers
	ScanMan.pas	ScanMan library module for Delphi
	VFinger.bas	VeriFinger library module for Visual Basic 6.0
	VFinger.vb	VeriFinger library module for Visual Basic .Net
Lib\		Directory with library files for Microsoft Visual C++ 6.0/7.0/7.1
	VFinger.dll.lib	VeriFinger DLL library file (for DLL version of VeriFinger SDK only)
	VFinger.NP.lib	VeriFinger not protected library file (for not protected library version of VeriFinger SDK only)
	ScanMan.dll.lib	ScanMan.dll library file
Res\		Directory with resources for sample applications
	Logo.bmp	Logo picture in bitmap format
	VFDemo.ico	Icon for the application
	VFDemoSmall.ico	Small icon for the application
	VF-Demo.exe.manifest	Manifest for the application
	VFDemo.mdb	Default database

`VFDemo.cpp\`	Directory with source of sample application for Microsoft Visual C++ 6.0/7.0/7.1
`VFDemo.dll.cpp\`	Directory with sample application for Microsoft Visual C++ 6.0/7.0/7.1 (for DLL version of VeriFinger SDK only)
`VFDemo.NP.lib.cpp\`	Directory with sample application for Microsoft Visual C++ 6.0/7.0/7.1 (for not protected lib version of VeriFinger SDK only)
`VFDemo.pas\`	Directory with sample application for Delphi 6
`VFDemo.bas\`	Directory with sample application for Visual Basic 6
`VFDemo.Net.vb\`	Directory with sample application for Visual Basic .Net
`VFDemo.Access`	Directory with sample application for Microsoft Access 2000
`VFDemo.java\`	Directory with sample application for Sun Java 2
`Install\`	Installation folder for HASP drivers, scanners, etc.
`Support\`	Contains SDK support information/tools: FAQ (Frequently Asked Questions), fingerprint features rescaling tool/DLL, etc. For more information please review `ReadMe.txt` file in `Support\` folder.
`Components\`	Directory with components (for VeriFinger SDK Extended only)
`VFDemoExt.asp\` `VFDemoExt.bas\` `VFDemoExt.cpp\`	Sample applications for ASP, Visual Basic 6.0, Visual C++ and HTML (for VeriFinger SDK Extended only)

VFDemoExt.html\	
License.html	VeriFinger SDK license
ReadMe.txt	ReadMe file for VeriFinger SDK
VeriFinger 4.2 SDK.pdf	This file
VeriFinger 4.2 SDK Extended.pdf	Components description (for VeriFinger SDK Extended only)

Later, where referenced:

`null` - `NULL` in C, `nil` in Delphi and `0` or `VF_DEFAULT_CONTEXT` in Basic and Java

`integer` - `INT` in C, `Integer` in Delphi and `Long` in Basic

`exception` - `exception` class in C, `Exception` class in Delphi and `Err` in Basic

Chapter 2. What's new

2.1. VeriFinger library

Version 4.2.0.3

- Features extraction with low quality image bug fix

Version 4.2.0.2

- Features generalization bug fix
- Minor bugfixes

Version 4.2.0.1

- Minor bugfixes

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 library.
- Added CrossMatch Verifier 300 scanner mode.
- VeriFinger can now return skeletonized image.

Version 4.1.0.0

- Completely new interface.
- 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. For more information see Parameters.
- 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. For more information see Parameters.

2.2. VeriFinger SDK

Version 4.2.0.3

- VeriFinger library updated to version 4.2.0.3
- Fixed VB.NET sample application project

Version 4.2.0.2

- VeriFinger library updated to version 4.2.0.2
- Fixed identification thread bug in C++ and Delphi sample applications

Version 4.2.0.1

- VeriFinger library updated to version 4.2.0.1
- Fixed XML parser header file in C++ sample application

Version 4.2.0.0

- Uses VeriFinger library version 4.2.0.0.

Version 4.1.0.0

- Uses VeriFinger library version 4.1.0.0.
- ScanMan library is 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.

2.3. VeriFinger SDK documentation

Version 4.2.0.3

- Reflects changes in VeriFinger SDK version 4.2.0.3

Version 4.2.0.2

- Reflects changes in VeriFinger SDK version 4.2.0.2

Version 4.2.0.1

- Reflects changes in VeriFinger SDK version 4.2.0.1
- Minor formatting updates

Version 4.2.0.0

- Based on VeriFinger SDK version 4.2.0.0.
- Added separate "What's new" sections for VeriFinger library, VeriFinger SDK and VeriFinger SDK documentation.

Chapter 3. 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.

Chapter 4. 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.

4.1. 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 wrapper functions do not return results code but throw exception (`VeriFingerException`) when error occurs. You can get error code using `getErrorCode` method.

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) ...
...
}
```

4.2. 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 quaity 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. To get error description in Java you also can use VeriFingerException class getMessage method.

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) ...
...
```

4.3. 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) {
}
```

4.3.1. 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;
```

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 wrapper throws exception.

4.3.2. 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 func-

tion. 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;
```

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 wrapper throws exception if error occurs.

4.3.3. 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;
```

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 wrapper throws exception if error occurs.

4.4. 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
}
```

4.4.1. 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;
```

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.

4.4.2. 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;
```

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.

4.5. 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);
```

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);
}
```

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)
}
```

4.5.1. 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;
```

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.

4.5.2. 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;
```

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.

4.6. 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 in-	2	`VF_TYPE_SHORT4`	SHORT	SmallInt	Integer/

teger					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.

Throught Java wrapper parameters must be passed converted to String type. Wrapper returns parameters also converted to String type.

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_NONE	0	No image is returned - the image will be the same as passed to features extraction function		
VF_RETURNED_IMAGE_BINARIZED	100	Binarized image will be returned (default)		
VF_RETURNED_IMAGE_SKELETONIZED	200	Skeletonized image will be returned		

Features matching (Verification and Identification)				
VFP_MATCHING_THRESHOLD	200		Integer	Minimal similarity of two features collections that are identical. Must be not less that zero. See also Matching threshold
VFP_MAXIMAL_ROTATION	201		Integer	Maximal rotation of two features collection to each other. Must be in range VFDIR_0..VFDIR_180. See also information about directions in Features and Matching details
VFP_MATCH_FEATURES	210		Integer	Obsolete, will be removed in the next version
VFP_MATCHING_SPEED	220		Integer	Speed of features matching. Can be one of the following:
VF_MATCHING_SPEED_LOW	0	Low matching speed		
VF_MATCHING_SPEED_HIGH	256	High matching speed		
Features generalization				
VFP_GENERALIZATION_THRESHOLD	300		Integer	Has the same meaning for features generalizatiom as VFP_MATCHING_THRESHOLD parameter for features matching. See also Matching threshold
VFP_GEN_MAXIMAL_ROTATION	201		Integer	Has the same meaning for features generalization as VF_MAXIMAL_ROTATION parameter for features matching

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_AFS2	800	Authentec AFS2		
VF_MODE_AUTHENTEC_AES4000	810	Authentec AES4000		

VF_MODE_ATMEL_FINGERCHIP	900	Atmel FingerChip
VF_MODE_BMF_BLP100	1000	BMF BLP100
VF_MODE_SECUGEN_HAMSTER	1100	SecuGen Hamster
VF_MODE_ETHENTICA	1200	Ethentica
VF_MODE_CROSSMATCH_VERIFIER300	1300	CrossMatch Verifier 300

4.6.1. 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;
```

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 wrapper throws exception 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", version, HIWORD(version), LOWORD(version));
        // Determine parameter VFP_MODE type
        vfp_mode_type = VFGetParameter(VFP_TYPE, (VOID*)VFP_MODE, NULL);
        // returned value: VF_TYPE_INT
        // Get integer parameter VFP_MODE value
        VFGetParameter(VFP_MODE, &mode, NULL);
        printf("Mode: %d", mode);
}
```

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);
```


4.6.2. 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;
```

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 Visu-

		al 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 wrapper throws exception 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);
}
```

4.6.3. Additional functions

The following functions are not part of VeriFinger library. For C they are implemented in `VFingerX.h` and `VFinger.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", version, HIWORD(version), LOWORD(version));
        // Determine parameter VFP_MODE type
        vfp_mode_type = VFGetParameterType(VFP_MODE);
        // returned value: VF_TYPE_INT
        // Get integer parameter VFP_MODE value
```

```
        mode = VFGetIntParameter(VFP_MODE);
        printf("Mode: %d", mode);
        // 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));
```

4.7. 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.

4.7.1. 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;
```

Parameters:

	width, Width	Fingerprint image width
	height, Height	Fingerprint image height
[in/out]	image, Image	Fingerprint image to extract features from. After execu-

		tion 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.

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);
}
```

4.8. 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.

4.8.1. 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;
```

Parameters:

	count, Count	Count of features collections to generalize
[in]	genFeatures,	Array of features collections to generalize

	GenFeatures	
[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.

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!"
}
```

4.9. 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.

4.9.1. 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;
```

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.

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));
}
```

4.10. 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");
```

4.10.1. 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;
```

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

.VeriFinger Java wrapper throws exception if error occurs.

4.10.2. 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;
```


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.

4.10.3. 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;
```

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.

4.11. 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;
```

4.12. 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];
        }
}
```

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, PVFMatchDetailsEx(@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]));
}
```

4.13. 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];
        }
};
```

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) ...
        ...
```


Chapter 5. 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).

5.1. Modules and supported scanners

In Light version of VeriFinger SDK ScanMan library contains no modules.

In Standart version of VeriFinger SDK ScanMan library contains these modules:

Module	Support for multiple scanners	Supported scanners
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

5.2. Library functions and error codes

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

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.3. 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.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.3.1. SMInitialize function

Initializes ScanMan library.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.3.2. SMFinalize function

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.4. Parameters

ScanMan library behavior is controlled through parameters. Parameters are retrieved and set by SMGetParameter and SMSetParameter functions.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.4.1. SMGetParameter function

Retrieves specified parameter value.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.4.2. SMSetParameter function

Sets specified parameter value.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.4.3. 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.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.5. Scanner enumeration

Application can determine number of connected scanners using SMGetScannerCount function and identifier of each connected scanner using SMGetScannerId function. You may also use SMGetScannerIds function to get list of connected scanners identifiers. Scanner identifier can be passed to capturing functions. See also Scanner identifiers.

5.5.1. SMGetScannerCount function

Retrieves number of connected scanners.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.5.2. 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.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.5.3. 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.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.6. Scanner monitoring

Application can use scanner monitoring to receive such event as scanner plugging and unplugging and scanner errors. 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:

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.6.1. SMSetMonitor function

Sets monitoring function

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.6.2. SMRemoveMonitor function

Removes monitoring function set by SMSetMonitor. So SMMonitorProc will not be more called.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.6.3. 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.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.7. 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:

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.7.1. 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.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.7.2. 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.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.7.3. 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.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.7.4. 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.

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.8. 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.

5.9. ScanMan Visual Basic support

Visual Basic 6.0/.Net is 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.

COM wrapper library exports two classes:

1. `FPScanner` - scanner class; one class instance represents one physical device.
2. `FPScannerMan` - scanners manager (enumerates, creates and monitor scanners).

`FPScannerMan` has following interface:

Methods	
Method	Description
`Initialize`	Initializes ScanMan library. Complete description is available in VeriFinger 4.2 SDK Documentation.
`Finalize`	Finalizes ScanMan library. Complete description is available in VeriFinger 4.2 SDK Documentation.
`EnumDeviceIDs`	Enumerates all connected and recognized devices. Complete description is available in VeriFinger 4.2 SDK Documentation.

`CreateScanner`	Creates scanner object for specified identificator. Complete description is available in VeriFinger 4.2 SDK Documentation.
`GetErrorDescription`	Returns description for specified error code. Complete description is available in VeriFinger 4.2 SDK Documentation.
Properties	
Property	Description
DeviceCount	Contains connected and recognized device count. Complete description is available in VeriFinger 4.2 SDK Documentation.
Events	
Event	Description
DeviceStatus	Event is raised when device status is changed (new scanner connected or existing disconnected, etc.). Complete description is available in VeriFinger 4.2 SDK Documentation.

`FPScanner` can be created only with CreateScanner method. It has such interface:

Methods

Method	Description
`StartCapturing`	Starts capturing fingerprint image from associated device. Complete description is available in VeriFinger 4.2 SDK Documentation.
`StopCapturing`	Stops capturing fingerprint image. Complete description is available in VeriFinger 4.2 SDK Documentation.
`WaitForImage`	Reads one fingerprint image from associated device. Functions will wait infinite time if TimeOut = 0. Complete description is available in VeriFinger 4.2 SDK Documentation.
`CaptureOneImage`	Starts capturing. Capturing will be automatically stopped after finger will be scanned or specified time will end. Complete description is available in VeriFinger 4.2 SDK Documentation.
`SaveImage`	Saves scanned image to specified file (BMP). Complete description is available in VeriFinger 4.2 SDK Documentation.
Properties	
Property	Description
DeviceID	Contains associated device identifier. Complete description is available in

	VeriFinger 4.2 SDK Documentation.
Events	
Event	Description
Error	Event is raised when error occurs. Complete description is available in VeriFinger 4.2 SDK Documentation.
Image	Passed scanned image to application. Complete description is available in VeriFinger 4.2 SDK Documentation.
Status	Event is raised when scanner status is changed. Complete description is available in VeriFinger 4.2 SDK Documentation.

Example

Complete description is available in VeriFinger 4.2 SDK Documentation.

5.10. ScanMan Java support

Java applications can access ScanMan library through JNI wrapper `SMJavaW.dll`. Library interface is declared in `ScanMan` class:

Category	Methods
Initialization	public static native void SMInitialize() throws ScanManException, Exception; Complete description is available in

	VeriFinger 4.2 SDK Documentation. public static native void SMFinalize() throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation.
Parameters	public static native String getParameter-Value(int parameter) throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation. public static native void setParameter-Value(int parameter, String value) throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation. public static native int getParameterType(int parameter) throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation.
Scanner enumeration	public static native int SMGetScannerCount() throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation. public static native String SMGetScan-nerId(int index) throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation.
Scanner monitoring	public static native void SMSetMonit-

	or(ScannersMonitor monitor) throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation. public static native void SMRemoveMonitor() throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation.
Capturing	public static native void SMStartCapturing(String id, ScannerEventListener listener) throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation. public static native void SMStopCapturing(String id) throws ScanManException, Exception; Complete description is available in VeriFinger 4.2 SDK Documentation.

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:

Complete description is available in VeriFinger 4.2 SDK Documentation.

Chapter 6. Sample applications

Sample applications are provided for Microsoft Visual C++ 6.0/7.0, Borland Delphi 6, Microsoft Visual Basic 6.0/.Net, Microsoft Access 2000 and Sun Java 2.

6.1. Microsoft Visual C++ 6.0/7.0/7.1

Source of Microsoft Visual C++ 6.0/7.0/7.1 (later referenced as C) application is located in `VFDemo.cpp` directory of SDK and `VFDemo.dll.cpp` directory (for DLL version of VeriFinger SDK) or `VFDemo.NP.lib.cpp` directory (for not protected library version of VeriFinger SDK). Project file for C is `VFDemo.dll.cpp.vcproj` (version 7.1), `VFDemo.dll.cpp.7.0.vcproj` (version 7.0) or `VFDemo.dll_cpp.dsp` (version 6.0) (`VFDemo.NP.lib.vcproj`, `VFDemo.NP.lib.7.0.vcproj` and `VFDemo.NP.lib.dsp` for not protected library version of VeriFinger SDK). Resource files are located in `Res` directory of SDK:

`Logo.bmp`	Logo picture in bitmap format
`VFDemo.ico`	Icon for the application
`VFDemoSmall.ico`	Small icon for the application
`VFDemo.exe.manifest`	Manifest for the application
`VFDemo.mdb`	Default database

Interfaces for VeriFinger library and ScanMan library are provided in `Include/` directory of SDK: `VFinger.h` and `ScanMan.h`. Demo application also uses library files in `Lib/` directory of SDK for these libraries (`VFinger.dll.lib` and `ScanMan.dll.lib`). For not protected library version of VeriFinger SDK library `VFinger.lib` file from the same directory is included into your C application project. Extended VeriFinger library and ScanMan library features are implemented in `VFingerX.h`, `VFingerX.cpp`, `ScanManX.h`, `ScanManX.cpp` files.

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 `COpenFile-Dialog` 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`).

VeriFinger library registration helper functions are defined in `VFRegister.h` and `VFRegister.cpp` files.

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, VeriFinger library and ScanMan library.

`VFRegisterForm.h` and `VFRegisterForm.cpp` files implement VeriFinger library registration dialog (`CRegisterForm`).

`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.

Main files for application are `VFDemo.dll.cpp` (`VFDemo.NP.lib.cpp` for not protected version of VeriFinger SDK). In `CVFDemoApp` class as a descendant of `CConsoleApplication` is defined and created in `WinMain` function. Both applications initialize VeriFinger and ScanMan libraries, 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 and VeriFinger and ScanMan libraries are displayed there. Then options and VeriFinger library 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 VeriFinger library 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 doesn't 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 OnModeIdentification	Mode»Identification command. Changes mode to vfdmIdentification (via 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
`ID_HELP_REGISTERENGINE` `OnHelpRegisterEngine`	Help»Register VeriFinger... command. Shows Register dialog where user registers VeriFinger library
`ID_HELP_CLEARENGINEREGISTRATION` `OnHelpClearEngineRegistration`	Help»Clear VeriFinger registration command. Clears registration information of VeriFinger library

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.

6.2. Borland Delphi 6

Source of application is located in `VFDemo.pas` directory. Project file - `VFDemo.pas.dpr`. Resource files are located in `Res/` directory of SDK:

`Logo.bmp`	Logo picture in bitmap format
`VFDemo.ico`	Icon for the application
`VFDemoSmall.ico`	Small icon for the application
`VFDemo.exe.manifest`	Manifest for the application
`VFDemo.mdb`	Default database

Interfaces for VeriFinger library and ScanMan library are provided in `Include/` directory of SDK: `VFinger.pas` and `ScanMan.pas` modules. Extended VeriFinger library and ScanMan library features are implemented in `VFinger.pas` and `ScanMan.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`).

VeriFinger library registration helper functions are defined in `VFRegister.pas` module.

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, VeriFinger library and ScanMan library.

`FormRegister.pas` and `FormRegister.dfm` modules implement VeriFinger library registration dialog (`TRegisterForm`).

`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`. Both applications initialize VeriFinger and ScanMan libraries, create main form and run it.

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 VeriFinger and ScanMan libraries are displayed there. Then options and VeriFinger library 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 dialog is shown if VeriFinger library is not registered and 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
`HelpRegisterExecute`	Help»Register VeriFinger... command. Shows Register dialog where user registers VeriFinger library
`HelpClearRegistrationExecute`	Help»Clear VeriFinger registration command. Clears registration information of VeriFinger library

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.

6.3. Microsoft Visual Basic 6.0/.Net

Source of Visual Basic 6.0 application is located in `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

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

Visual Basic .Net Forms	
File	Description
MainForm.vb MainForm.resX	Main form of sample application. Contains main menu, two picture boxes (for original and binary fingerprint image), log window and progress bar
DialogSettings.vb DialogSettings.resX	Application settings form. Allows change similarity threshold by changing FAR (false acceptance rate) and image resolution (DPI)
ScannerList.vb ScannerList.resX	Scanners list form. Displays list of scanners and allow choose one
DialogRegister.vb DialogRegister.resX	VeriFinger DLL registration form
VeriFinger.ico	VeriFinger icon file
Visual Basic .Net Modules	
File	Description

VFinger.vb	VeriFinger interface declaration and implementation of helper functions
Service.vb	Useful collections of functions/sub for fingerprint features drawing and image manipulations
DataBase.vb	Operations with database
Settings.vb	Application settings
Visual Basic 6.0 Class Modules	
File	Description
FingerprintRecord.vb	Class which defines 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.sln`	Visual Studio Solution File
`VFingerVBProject.vbproj`	Visual Basic Project
`AssemblyInfo.vb`	Assembly information

It is impossible to use VeriFinger DLL directly from Visual Basic 6.0 or .Net therefore special parser was written. VeriFinger Visual Basic Parser DLL source is located in `Wrappers` directory. Same parser is used in Visual Basic .Net sample.

Visual Basic 6.0 and .Net sample applications 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.

6.3.1. 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();
INT WINAPI VBVFGenerateId(CHAR * serial, CHAR * id);
INT WINAPI VBVFRegister(CHAR * serial, CHAR * key);
// 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);
```

6.3.2. Usage of VeriFinger Visual Basic Parser

VeriFinger Parser DLL exports following functions, wrapping VeriFinger functions:

Function	Description
`VBVFInitialize`	For more information please see VFInitialize
`VBVFFinalize`	For more information please see VFFinalize
`VBVFRegistration-Type`	For more information please see VFRegistrationType
`VBVFGenerateId`	For more information please see VFGenerateId
`VBVFRegister`	For more information please see VFRegister
`VBVFCreateContext`	For more information please see VFCreateContext
`VBVFFreeContext`	For more information please see VFFreeContext
`VBVFGetParameter-Type`	For more information please see Additional parameter-related functions
`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 Lib "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) and `VFinger.vb` module (Visual Basic .Net sample).

6.4. Microsoft Access 2000

Microsoft Access 2000 sample is located in `VFDemo.access\` subdirectory of SDK. Directory contains:

File	Description
`VF-Demo_msaccess.mdb`	VeriFinger Sample
`VFinger.dll`	VeriFinger DLL
`VFVBP42.dll`	VeriFinger Parser DLL

Code and database are located in the same file (database - `VFDemo_msaccess.mdb`). MS Access uses same VeriFinger parser as Visual Basic samples (`VFVB41.DLL`). Sample contains modules and forms that are very similar to Visual Basic 6 modules and forms:

VBA Forms	
Form	Description
VFDemo	Main form of sample. Contains two picture boxes (for original and binary fingerprint image), log window and progress bar

Settings	Sample settings form. Allows change similarity threshold by changing FAR (false acceptance rate) and image resolution (DPI)
Scanners	Displays list of scanners and allow choose one
VFRegister	VeriFinger DLL registration form
VBA Modules	
Module	Description
VFinger	VeriFinger interface declaration and implementation of helper functions
Service	Useful collections of functions/sub for fingerprint features for image manipulations
DataBase	Operations with database
Registration	Operations and global variables for VeriFinger DLL registration
Scanners	Declares global scanners variables
Settings	Operations and global variables for sample settings
Bitmaps	Saving to bitmap implementation
VBA Class Modules	
File	Description
FingerprintRecord	Class which defines fingerprint record

To run sample open VFDemo form. Access sample is very similar to Visual Basic samples. Here

are main differences:

- Sample work is controlled by controls which are on form (Visual Basic samples are controlled using menu)
- Fingerprint features are drawn using parser `DrawFingerprintFeatures` function.

Please copy `VFinger.dll` and `VFVBP41.dll` to Windows System directory before running sample. It is also recommended copy all scanners support files into Windows `System` directory. `sslCom.dll` must be registered before running this sample. Use `bin\sslCOMreg.bat` to register this dll.

6.5. 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 `VFDemo.java\` SDK subdirectory. It contains:

Directory/File	Description

`Application\`	Contains demo and it's source
`Wrappers\`	Contains VeriFinger and ScanMan libraries wrappers 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.

