VeriLook 2.0 SDK

VeriLook 2.0 SDK

Table of Contents

Chapter 1. Introduction

VeriLook SDK consists of VeriLook library and sample applications:

- VeriLook library is a face recognition engine.
- Sample applications provide an example of image acquisition and VeriLook SDK usage.

VeriLook library requires face images to meet certain constraints to ensure optimal face recognition performance.

VeriLook SDK can be used in a number of compilers under any of these operating systems: Windows Me and Windows NT/2000/XP. VeriLook library is provided as Win32 dynamic link library (DLL) `VLook.dll` (in `Bin\` subdirectory). VeriLook library can be provided as .lib file to use in Microsoft Visual C++ 6.0/7.0 (in `Lib\` subdirectory).

Library interface and sample applications are available for the following compilers:

- Microsoft Visual C++ 6.0/7.0 (later referenced as C). Header file `VLook.Interface.h` (in `Include\` subdirectory) proceeded by `Sys.h` (in `Common\` subdirectory) and neurotec namespace usage declaration (`using namespace neurotec`), DLL library file `VLook.lib` (for .lib version of VeriLook library `VLook.lib` file in Lib\ subdirectory) 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 or higher you may have to create another `VLook.lib` file. For example for Borland C/C++ compiler you should 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 the `VLook.Interface.h` and types defined in the `Types.Base.h`. Sample application is available in `VLDemo.cpp.dll\` (for .lib version of VeriLook library – in `VLDemo.cpp.lib\` subdirectory) subdirectory.
- Microsoft Visual C# .NET. DLL library file VLook.dll, VLDemo.mdb have to be in your working directory.
- Microsoft Visual Basic 6.0 (further referenced as Visual Basic). Module `VLook.bas` (in `Include\` subdirectory) has to be included to your application project. This module is using VeriLook parser `VLVBP.DLL` (in `VLDemo.bas\` subdirectory). In demo program (`VLDemo.bas\` subdirectory) you can find `Service.bas` and `DataBase.bas` files, which can be helpful developing your application(s).
- Microsoft Access 2000 (further referenced as Access). Sample application and face database are located in the same file (database). Access demo utilizes the same VeriLook parser as Visual Basic sample (`VLVBP.DLL`). Sample contains modules that are very similar to Visual Basic modules. VeriLook functions are also declared in the same way as in Visual Basic sample therefore only differences will be noticed in this documentation.

SDK directory contains:

`Bin\`	Subdirectory with binaries.	
	`VLook.dll`	VeriLook library.
	`VLDemo.cpp.dll.exe`	Visual C++ sample application (using VeriLook.dll).
	`VLDemo.cpp.lib.exe`	Visual C++ sample application (using VeriLook.lib)
	`VLCapturer.dll`	ActiveX for capturing images from webcam (used by Visual Basic samples).
	`VLCapturerReg.bat`	Command file to register `VLCapturer.dll`.
`Common\`	Subdirectory with header and implementation files of common usage.	
	`Sys.h` and `Sys.cpp`	Basic operations.
	`Types.Base.h` and `Types.h`	Basic types.
	`SImage.h` and `SImage.cpp`	Image class.
`Install\HASP\`	HASP device driver installation.	
`Include\`	Subdirectory with header files	
	`VLook.Interface.h`	VeriLook library header for C/C++ compilers
`Lib\`	DLL library files for Visual C++.	
	`VLook.dll.lib` or `VLook.lib` and `VLook.VC6.lib`	VeriLook DLL library file or VeriLook library file (for .dll and .lib versions of VeriLook library accordingly)
`Res\`	Sample applications resource files.	
	`Neurotec.ico`	Icon for the application.
	`Neurotec.small.ico`	Small icon for the application.
	`VLDemo.mdb`	Default database.
`VLDemo.cpp\`	Visual C++ sample application source.	

`VLDemo.NET\`	Visual C# sample application, wrapper, showing control source.	
`VeriLookDemo`	VeriLook C# sample application source.	
`VLWrapperNET`	VeriLook C# wrapper source.	
`VideoControl`	Visual C# image showing control source.	
`VLDemo.bas\`	Visual Basic sample application source.	
`Parser`	VeriLook Visual Basic wrapper source.	
`VLCapturer`	ActiveX component for image capturing from webcam source.	
`VL-Demo.Access\`	Access (VBA) sample application source.	
	`License.html`	License file.
	`ReadMe.txt`	ReadMe file.
	`VeriLook 2.0 SDK.pdf`	PDF help file.
	`VeriLook 2.0 SDK.chm`	CHM help file.

Later, where referenced:

- null - `NULL` in C and 0 or `VL_DEFAULT_CONTEXT` in Visual Basic
- integer – `INT` in C, `Long` in Visual Basic

Chapter 2. What's New

Version 2.0.0.1

- Improved matching speed.
- New sample application - Visual C#.

Version 2.0.0.0

- Improved reliability.
- Improved matching speed. Matching is twice faster now.
- Changed template. New template occupies 2.9KB and is not compatible with templates of previous versions.

Version 1.1.0.0

- Improved quality of face localization algorithm.
- Multiple faces can be localized in the same amount of time now.
- Added demonstration of multiple face localization to Visual C++ sample application.

Version 1.0.1.4

- CHM file added to documentation.

Version 1.0.1.3

- New sample application - Access.

Version 1.0.1.2

- New sample application - Visual Basic.

Version 1.0.1.1

- Removed unused files from distribution.
- Updated and cleaned-up documentation.

Version 1.0.1.0

- Added ReadMe.txt.
- Added HASP Driver Installation.
- Removed unused files from distribution.
- Updated and cleaned-up documentation.
- Demo: Added configurable ranking list size.

Version 1.0.0.1

- Initial release.

Chapter 3. Requirements

3.1. Demo application requirements

- 128 MB of RAM, 1Ghz CPU, 2MB HDD space for the installation package.
- Microsoft Windows 98/Me/NT/2000/XP.
- DirectX 8.1 or later. You can download DirectX upgrade from Microsoft web site
- Microsoft GDI+. This library is supplied with Windows XP and Windows .NET Server family. If you are using any other modern Windows platform (Windows 98/Me and Windows NT 4.0/2000) you should download and install it from Microsoft web site.
- The Microsoft® XML Parser (MSXML) 3.0 SP4 is required so if it is not already in the system you should download and install it from Microsoft web site.
- Optionally, video capture device (web camera).

Chapter 4. Face image constraints

Face recognition is very sensitive to image quality so maximum care should be attributed to image acquisition.

4.1. Pose

The frontal pose (full-face) must be used. Rotation of the head must be less than ±5 degrees from frontal in every direction - nodded up/down, rotated left/right, tilted right/left.

4.2. Expression

The expression should be neutral (non-smiling) with both eyes open, and mouth closed. Every effort should be made to have supplied images comply with this specification. A smile with closed jaw is allowed but not recommended.

4.2.1. Examples of Non-Recommended Expressions

- A smile where the inside of the mouth is exposed (jaw open).
- Raised eyebrows.
- Closed eyes.
- Eyes looking away from the camera.
- Squinting.
- Frowning.
- Hair covering eyes.
- Rim of glasses covering part of the eye.

4.3. Face changes

Beard, moustache and other changeable face features influence face recognition quality and if frequent face changes are typical for some individual, face database should contain e.g. face with beard and cleanly shaved face enrolled with identical ID.

4.4. Lighting

Lighting must be equally distributed on each side of the face and from top to bottom. There should be no significant direction of the light or visible shadows. Care must be taken to avoid "hot spots". These artifacts are typically caused when one, high intensity, focused light source is used for illumination.

4.5. Eyeglasses

There should be no lighting artifacts on eyeglasses. This can typically be achieved by increasing the angle between the lighting, subject and camera to 45 degrees or more. If lighting reflections cannot be removed, then the glasses themselves should be removed. (However this is not recommended as face recognition typically works best when matching people with eyeglasses against themselves wearing the same eyeglasses).

Glasses have to be of clear glass and transparent so the eyes and irises are clearly visible. Heavily tinted glasses are not acceptable.

4.6. Web cameras

As web cameras are becoming one of the most common personal video capturing devices, we have conducted small video image quality check. Most of cheap devices tend to provide 320x240 images of low quality, insufficient for biometrical use, however few of higher quality devices deserve a mention: Logitech Quick Cam Pro 4000, Logitech Quick Cam Zoom, and Creative Pro Ex. As a general rule, true 640x480 resolution (without interpolation) and a known brand name are recommended.

Images should be enrolled and matched using the same camera, as devices have different optical distortions that can influence face recognition performance.

Chapter 5. VeriLook library

VeriLook library is a face recognition engine that you can use in applications developed by you.

VeriLook library enables application to implement such scenarios as user enrollment, user verification and user identification using face images. It provides a number of functions to implement such behavior.

When enrolling a user application can use face detection and features extraction functions that extracts features from face image (for more information see Face 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 a face image are compared with template features that are in the database or in some other location. See Verification.

VeriLook 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.

VeriLook library behavior is controlled through parameters.

5.1. Library functions

VeriLook library contains the following functions grouped by categories:

Registration	
VLRegistrationType	Returns registration type of VeriLook library
VLGenerateId	Generates registration id from serial number
VLRegister	Registers VeriLook library
Initialization	
VLInitialize	Initializes VeriLook library
VLFinalize	Un-initializes VeriLook library
Contexts	
VLCreateContext	Creates a context
VLFreeContext	Deletes the context
Parameters	
VLGetParameter	Retrieves parameter value

VLSetParameter	Sets parameter value
Face Detection	
VLDetectFaceOnce	Detects the face in a single image.
VLDetectMultipleFacesOnce	Detects multiple faces in a single image.
VLDetectFace	Used to detect the face in a sequence of images.
Features extraction	
VLExtract	Extracts features from face image
Features generalization	
VLGeneralize	Generalizes multiple feature collections to single feature collection
Verification	
VLVerify	Matches two feature collections

Each function (except for registration, initialization and contexts functions) takes last argument of type `void*`. It is the context in which VeriLook library functions are called. Pass null to use default context. For more information see Initialization and Contexts.

Each of these functions (except for the VLCreateContext) returns integer value to indicate result of the execution. If it is less than zero then execution of the function failed and the value indicates error code.

You can use `VLFailed` and `VLSucceeded` functions to determine if the execution of the function failed or succeeded:

C++:

```
#define VLFailed(result) …
#define VLSucceeded(result) …
```

Visual Basic:

```
Public Function VLSucceeded(ByVal result As Long) As Boolean
Public Function VLFailed(ByVal result As Long) As Boolean
```

5.2. Error codes

The following error codes are defined:

General		
`VLE_OK`	0	OK, no error
`VLE_FAILED`	-1	Failed
`VLE_OUT_OF_MEMORY`	-2	Out of memory
`VLE_NOT_INITIALIZED`	-3	VeriLook library is not initialized
`VLE_ARGUMENT_NULL`	-4	One of the required function arguments is `null`
`VLE_INVALID_ARGUMENT`	-5	One of the function arguments has an invalid value
`VLE_INVALID_TEMPLATE`	-6	Template is not generated by VeriLook
`VLE_TEMPLATES_NOT_COMPATIBLE`	-7	Versions templates are not compatible
`VLE_NOT_IMPLEMENTED`	-9	Function is not implemented
Registration		
`VLE_NOT_REGISTERED`	-2000	VeriLook library is not registered
`VLE_INVALID_SERIAL_NUMBER`	-2001	Specified serial number is invalid
`VLE_INVALID_REGISTRATION_KEY`	-2002	Specified registration key is invalid
`VLE_SCANNER_DRIVER_ERROR`	-2003	Scanner driver error
`VLE_REGISTRATION_NOT_NEEDED`	-2004	No need to register VeriLook library
`VLE_NO_SCANNER`	-2005	No scanner found
`VLE_MORE_THAN_ONE_SCANNER`	-2006	More than one scanner found
`VLE_LM_CONNECTION_ERROR`	-2007	Error communicating with License Manager
`VLE_LM_NO_MORE_LICENCES`	-2008	No more License Manager licenses are available

Parameters		
`VLE_INVALID_PARAMETER`	-10	Parameter identifier is invalid (unknown)
`VLE_PARAMETER_READ_ONLY`	-11	Parameter is read only
Features extraction		
`VLE_ILLEGAL_IMAGE_RESOLUTION`	-101	Specified image resolution is illegal
`VLE_ILLEGAL_IMAGE_SIZE`	-102	Specified image size is illegal
`VLE_LOW_QUALITY_IMAGE`	-103	Warning. Image quality is low
VeriLook specific		
`VLE_INVALID_MODE`	-1000	Function called in invalid mode

5.3. Registration

You have to register VeriLook library before using it. If library is not registered all functions (except for initialization, contexts and parameters functions) will return `VLE_NOT_REGISTERED`. There are several registration types available: not protected library, registration with HASP key, registration to PC, registration to UareU scanner (for current VeriFinger customers) 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, plug it to LPT or USB port and call VLRegister function, pass serial number provided with your VeriLook library license and `"HASP"` as registration key to it.

If you are using registration to PC then call VLGenerateId function and pass serial number provided with your VeriLook library license to it. 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 VLRegister function.

If you are using registration to UareU scanner then call VLGenerateId function and pass serial number provided with your VeriLook library license to it. 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 VLRegister 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 VeriLook library is registered (and if it needs registration at all) call VLRegistrationType function.

Example:

C++:

```
// Registration to PC
// Your serial number here
Char serial_number[] = "xxxx-xxxx-xxxx-xxxx";
// Registration id generation
{
        Char registration_id[100];
        VLGenerateId(serial_number, registration_id);
}
// Received registration key
Char registration_key[] = "xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx";
// Register VeriLook library
{
        VLRegister(serial_number, registration_key);
}
```

Visual Basic:

```
' Registration to PC
' 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 = VLGenerateId(SerialNumber, RegistrationId)
' Received registration key
Dim RegistrationKey As String
RegistrationKey = "xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx"
' Register VeriLook library
ErrCode = VLRegister(SerialNumber, RegistrationKey)
```

C#:

```
//VeriLook VLN = new VeriLook();
// Registration to PC
// Your serial number here
string serialNumber = "xxxx-xxxx-xxxx-xxxx";
// Registration id generation
```

```
{
        string registrationID;
        registrationID = VLN.GenerateId(serialNumber);
}
// Received registration key
string registrationKey = "xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx-xxxx";
// Register VeriLook library
{
        VLRegister(serial_number, registrationKey);
}
```

5.3.1. VLRegistrationType function

Returns VeriLook library registration type.

C++:

```
Int VLOOK_API VLRegistrationType();
```

Visual Basic:

```
Public Declare Function  VLRegistrationType _
        Lib "VLVBP.DLL" Alias "VBVLRegistrationType" _
        () As Long
```

C#:

```
public int RegistrationType{get;}
```

Return values: Returns one of the following values:

VL_RT_NOT_PROTECTED	0	VeriLook library is not protected. No need to register
VL_RT_HASP	1	HASP key found either on LPT or USB port. No need to register
VL_RT_PC	2	VeriLook library is registered to PC
VL_RT_UAREU	4	VeriLook library is registered to U.are.U scanner

VL_RT_UNREGISTERED	6	VeriLook library is not registered. Call VLRegister function to register
VL_RT_LAN	8	VeriLook library is registered in License Manager on LAN

5.3.2. VLGenerateId function

Generates registration id from specified serial number. Serial number and registration id have to be arrays of characters (strings) pointers to first element of each have to be passed to the function. Array for registration id has to be large enough to store the string (100 characters is enough).

C++:

```
Int VLOOK_API VLGenerateId(Char * serial, Char * id);
```

Visual Basic:

```
Public Declare Function VLGenerateId _
        Lib "VLVBP.DLL" Alias "VBVLGenerateId" _
        (ByVal Serial As String, ByVal id As String) As Long
```

C#:

```
public string GenerateId(string serial);
```

Parameters:

[in]	Serial	Serial number of VeriLook library license
[out]	Id	After execution of the function contains registration id for the serial number

Return values: If serial number or registration id is `null` returns `VLE_ARGUMENT_NULL`.If serial number is invalid returns `VLE_INVALID_SERIAL_NUMBER`.Otherwise generates registration id and returns `VLE_OK` (in case of error returns `VLE_FAILED`).

5.3.3. VLRegister function

Registers VeriLook 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 VLOOK_API VLRegister(Char * serial, Char * key);
```

Visual Basic:

```
Public Declare Function VLRegister _
        Lib "VLVBP.DLL" Alias "VBVLRegister" _
        (ByVal Serial As String, ByVal Key As String) As Long
```

C#:

```
public void Register(string serial, string key);
```

Parameters:

[in]	Serial	Serial number of VeriLook library license.
[in]	Key	Registration key for serial number and registration id (received from Neurotechnologija or its distributor).

Return values: If VeriLook library is not protected, returns `VLE_REGISTRATION_NOT_NEEDED`.If serial number or registration key is `null` returns `VLE_ARGUMENT_NULL`.If serial number is invalid returns `VLE_INVALID_SERIAL_NUMBER`.If registration key is invalid returns `VLE_INVALID_REGISTRATION_KEY`.Otherwise registers VeriLook library and returns `VLE_OK` (in case of error returns `VLE_FAILED`).

5.4. Initialization

VeriLook library requires initialization to be performed before any function call and un-initialization to be performed after all functions call. This is performed using VLInitialize and VLFinalize functions.

Each successful call to VLInitialize should have a corresponding call to VLFinalize. So you can call VLInitialize more than one time, but you have to call VLFinalize equal number of times.

Example:

C++:

```
// Main application function
{
        // Application initialization code
        VLInitialize();
        // Other application code
        VLFinalize();
        // Application un-initialization code
}
```

Visual Basic:

```
' In project source which is using Main sub as startup object
Sub Main()
        ' Application initialization code
        VLInitialize
        ' Other application code
        VLFinalize
        ' Application uninitialization code
End Sub

' In project source which is using form as startup object
Private Sub Form_Load…
        VLInitialize
        ' Application initialization code
End Sub

' Other application code
Private Sub Form_Unload…
        ' Application uninitialization code
        VLFinalize
End Sub
```

5.4.1. VLInitialize function

Creates default context by calling VLCreateContext function and initializes VeriLook library.

C++:

```
Int VLOOK_API VLInitialize();
```

Visual Basic:

```
Public Declare Function VLInitialize _
        Lib "VLVBP.DLL" Alias "VBVLInitialize" _
        () As Long
```

Return values: If succeeded return value indicates number of times function has been called before. If it first call to the function return value will be zero.If default context is not created VLE_OUT_OF_MEMORY is returned.

5.4.2. VLFinalize function

Destroys default context by calling VLFreeContext function and un-initializes VeriLook library if call to the function corresponds to first call to VLInitialize function.

C++:

```
Int VLOOK_API VLFinalize();
```

Visual Basic:

```
Public Declare Function VLFinalize _
        Lib "VLVBP.DLL" Alias "VBVLFinalize" _
        () As Long
```

Return values: Return value indicates number of times function should be more called (number of VLInitialize calls without VLFinalize calls).If VeriLook library was not initialized returns VLE_NOT_INITIALIZED.

5.5. Contexts

Context is a set of parameters and internal structures that VeriLook library functions use. They are created with VLCreateContext function and destroyed with VLFreeContext function.

Contexts enable different application parts to work with VeriLook library simultaneously. Inside one context no VeriLook functions should be called simultaneously because they are not guaranteed to be thread-safe. VeriLook 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 also to use different parameters in different situations. For example you can perform face detection and features extraction for different cameras in different contexts with different set of parameters. For more information see Parameters.

Example: Working from different threads:

C++:

```
// First thread function
```

```
{
        // Create context
        HVLCONTEXT context = VLCreateContext();
        // Call VeriLook library functions, for example
        VLVerify(…, context);
        // Delete context
        VLFreeContext(context);
}

// Second thread function
{
        // Create context
        HVLCONTEXT context = VLCreateContext();
        // Call VeriLook library functions, for example
        VLVerify(…, context);
        // Delete context
        VLFreeContext(context);
}
```

Visual Basic:

```
' First thread code
Dim ErrCode as Long
Dim Context as Long
' Create context
Context = VLCreateContext()
' Call VeriLook library functions, for example
ErrCode = VLVerify(…, Context)
' Delete context
ErrCode = VLFreeContext(Context)

' Second thread code
Dim ErrCode as Long
Dim Context as Long
Context = VLCreateContext()
' Call VeriLook library functions, for example
ErrCode = VLVerify(…, Context)
' Delete context
ErrCode = VLFreeContext(Context)
```

Example: Contexts with different parameters:

C++:

```
void * context1; // First context
void * context2; // Second context

// Initialization function
```

```
{
        // Set parameters for default context
        VLSetParameter(…, NULL);
        VLSetParameter(…, NULL);
        // Create first context
        context1 = VLCreateContext();
        // Set parameters for first context
        VLSetParameter(…, context1);
        VLSetParameter(…, context1);
        // Create second context
        context2 = VLCreateContext();
        // Set parameters for second context
        VLSetParameter(…, context2);
        VLSetParameter(…, context2);
}

// Some application function
{
        void * context;
        if (/* image from first camera */)
                context = context1;
        else if(/* image from second camera */)
                context = context2;
        else
                context = NULL; // default context
        // Call VeriLook library functions, for example
        VLExtract(…, context);
}

// Uninitialization function
{
        // Delete first context
        VLFreeContext(context1);
        // Delete second context
        VLFreeContext(context2);
}
```

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 = VLSetParameter(…, VL_DEFAULT_CONTEXT)
ErrCode = VLSetParameter(…, VL_DEFAULT_CONTEXT)
' Create first context
```

```
context1 = VLCreateContext()
' Set parameters for first context
ErrCode = VLSetParameter(…, context1)
ErrCode = VLSetParameter(…, context1)
' Create second context
context2 = VLCreateContext()
' Set parameters for second context
ErrCode = VLSetParameter(…, context2)
ErrCode = VLSetParameter(…, context2)

' Some application function

Dim context as long
if ' image from first camera
        context = context1;
else
        if ' image from second camera
                context = context2
        else
                context = VL_DEFAULT_CONTEXT ' default context
        end if
end if
' Call VeriLook library functions, for example
ErrCode = VLExtract(…, context)

' Uninitialization function

' Delete first context
ErrCode = VLFreeContext(context1)
' Delete second context
ErrCode = VLFreeContext(context2)
```

5.5.1. VLCreateContext function

Creates context with default parameters.

C++:

```
void * VLOOK_API VLCreateContext();
```

Visual Basic:

```
Public Declare Function VLCreateContext _
        Lib "VLVBP.DLL" Alias "VBVLCreateContext" _
        () As Long
```

Return values: Return value is newly created context.If context cannot be created returns `VLE_OUT_OF_MEMORY`.

5.5.2. VLFreeContext function

Deletes context created with VLCreateContext.

C++:

```
Int VLOOK_API VLFreeContext(void* context);
```

Visual Basic:

```
Public Declare Function VLFreeContext _
        Lib "VLVBP.DLL" Alias "VBVLFreeContext" _
        (ByVal context As Long) As Long
```

Parameters:

[in]	context	Context to delete

Return values: If context is `null` returns `VLE_ARGUMENT_NULL` else returns `VLE_OK`.

5.6. Parameters

Some VeriLook algorithm aspects are controlled through parameters. Parameters are retrieved and set for the specified context by VLGetParameter and VLSetParameter functions. Some parameters are read only (informational). If you will try to set a read only parameter VLSetParameter function will return `VLE_PARAMETER_READ_ONLY`. If you will pass an invalid parameter name to one of these functions it will return `VLE_INVALID_PARAMETER`.

Parameters can be of the following types:

Referenced as	Size (bytes)	VL_TYPE_XXX constant	C equivalent	Basic equival-ent
Double word	4	VL_TYPE_DWORD	DWORD	Long
Integer	4	VL_TYPE_INT	INT	Long
Boolean	4	VL_TYPE_BOOL	BOOL	Boolean
String	4	VL_TYPE_STRING	CHAR*	String

To determine parameter type call VLGetParameter function with `".GetType"` following the parameter name (see example for VLGetParameter function). Instead of the value the type of parameter will be returned as `VL_TYPE_XXX` constant.

When retrieving a parameter value pass pointer to variable of parameter type as value for VL-GetParameter 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.

Parameter value to set must be stored in the 32bit variable (double word) rather than simply cast to it. For convenience, `VLook.Interface.h` file contains type conversion helper macros/templates:

'C' - `TO_DWORD(value)`, `FROM_DWORD(type, value)`

'C++' - `DWordToDWord(T value)`, `T FromDWord(DWord value)`

The following parameter names are defined:

Identifier	Read only	Type	Description
`libraryName`	x	String	Name of the VeriLook library
`versionHigh`	x	Double word	Major version of VeriLook library
`versionLow`	x	Double word	Minor version of VeriLook library
`copyright`	x	String	Copyright of VeriLook library
`featuresSize`	x	Integer	Specifies size of features in bytes.
`DetectFace.minIOD`		Integer	Minimal interocular distance of faces. Faces with smaller interocular distance won't be detected by face detection functions.
`DetectFace.maxIOD`		Integer	Maximal interocular distance of faces. Faces with larger interocular distance won't be detected by face detection functions.
`DetectFace.similarityThreshold`		Float	Threshold for face detection functions. Higher threshold can be used to detect faces more rarely but more constrained and more suitable for face verification.

`DetectFace.maxFaces`		Integer	Maximal amount of faces to search in a single image.
`Detect-Face.detectEyesOnFaces`		Integer	Maximal number of faces to detect precise positions of eyes in.
`Detect-Face.symmetryThreshold`		Float	Threshold for face detection functions. Lower threshold can be used to detect faces more rarely but more constrained and more suitable for face verification.
`Detect-Face.maxFramesCount`		Integer	This threshold is used only for detecting faces in a sequence of images. This is maximum amount of frames to search for face. If face was not localized in none of these frames, face detection in a sequence of images fails.

5.6.1. VLGetParameter function

Retrieves specified parameter value for the specified context.

C++:

```
Int VLOOK_API VLGetParameter(Char * parameter, void * value, void * context);
```

Visual Basic:

```
Public Declare Function VLGetParameter _
        Lib "VLVBP.DLL" Alias "VBVLGetParameter" _
        (ByVal Parameter As String, ByRef value As Variant, _
         ByVal context As Long) As Long
```

C#:

```
public object GetParameter(string parameter);
```

Parameters:

[in]	parameter	Parameter identifier to retrieve
[out]	value	Pointer to variable that will receive parameter value.

[in]	context	Context to retrieve parameter from. NULL for default context

Return values: If context is `null` and VeriLook library is not initialized returns `VLE_NOT_INITIALIZED`.If parameter is invalid (unknown) returns `VLE_INVALID_PARAMETER`.If value is `null` returns `VLE_ARGUMENT_NULL`. For string parameters returns length of the string (not including the terminating null character).Otherwise returns `VLE_OK`.

Example:

C++:

```
// Get VeriLook library name
Char* libraryName = NULL;
Int paramType;
VLGetParameter("libraryName.GetType", &paramType, NULL);
Int size = VLGetParameter("libraryName", NULL, NULL);
libraryName = new Char[size];
VLGetParameter("libraryName", (void*) libraryName, NULL);
delete [] libraryName;

// Get VeriLook library version
DWord versionLow, versionHigh;
VLGetParameter("versionLow", (void*) &versionLow, NULL);
VLGetParameter("versionHigh", (void*) &versionHigh, NULL);
printf("Version: %u.%u", version, versionLow, versionHigh);
```

Visual Basic:

```
' Some application function/sub
Dim Name As Variant
Dim Version As Variant
' Get VeriLook library name
VLGetParameter "libraryName", Name, VL_DEFAULT_CONTEXT
MsgBox Name
' Get VeriLook library major version
VLGetParameter "versionHigh", Version, VL_DEFAULT_CONTEXT
MsgBox "Version: " & CStr((Version And &HFFFF0000) / &H10000) & "." _
        & CStr(Version And 65535)
```

C#:

```
VeriLook VLN = new VeriLook();
//get maximum amount of frames
try{
  int maximumFramesCount;
  maximumFramesCount = (int)VLN.GetParameter(VeriLook.MaxFramesCount);
}catch(VeriLookException ex1)
{...}
  catch(ArgumentException ex2)
{...}


//get VeriLook library version
string libraryName;
libraryName = (string)VLN.GetParameter(VeriLook.LibraryName);

//get features size
int featuresSize;
featuresSize = (int)VLN.GetParameter(VeriLook.FeaturesSize);

//get threshold for face detection
float threshold;
threshold =
        (float)VLN.GetParameter(VeriLook.DetectFaceSimilarityThreshold);

//get versionHigh
string verHigh;
verHigh = (string)VLN.GetParameter(VeriLook.VersionLow);
```

5.6.2. VLSetParameter function

Sets specified parameter value for the specified context.

C++:

```
Int VLOOK_API VLSetParameter(Char * parameter, DWord value, void * context);
```

Visual Basic:

```
Public Declare Function VLSetParameter _
        Lib "VLVBP.DLL" Alias "VBVLSetParameter" _
        (ByVal Parameter As String, ByVal value As Variant, _
         ByVal context As Long) As Long
```

C#:

```
public void SetParameter(
        string parameter,
        object parValue
)
```

Parameters:

[in]	Parameter	Parameter identifier to set
[out]	Value	Parameter value to set
[in]	Context	Context to set parameter to. Null for default context

Return values: If context is `null` and VeriLook library is not initialized returns `VLE_NOT_INITIALIZED`.If identification is started returns `VLE_INVALID_MODE`.If parameter is invalid returns `VLE_INVALID_PARAMETER`.Otherwise returns `VLE_OK`.

Example:

C++:

```
Float DetectFaceSimilarityThreshold = 132.43f;
VLSetParameter("DetectFace.similarityThreshold",
        ToDWord(DetectFaceSimilarityThreshold), NULL);
```

Visual Basic:

```
Dim DetectFaceSimilarityThreshold as Variant
DetectFaceSimilarityThreshold = CSng(132.43)
VLSetParameter "DetectFace.similarityThreshold", _
        DetectFaceSimilarityThreshold, VL_DEFAULT_CONTEXT
```

C#:

```
VeriLook VLN = new VeriLook();
VLN.SetParameter(VeriLook.DetectFaceSimilarityThreshold, 132.43f);
int maxFramesCount = 10;
VLN.SetParameter(VeriLook.DetectFaceMaxFramesCount, maxFramesCount);
```

5.7. Image representation

Any image used as an argument for VeriLook functions is of `ByteImage` structure type that is declared in a header `Types.Base.h`.

C++:

```
/* this structure is already defined in "Types.Base.h"
typedef struct _ByteImage
{
        UInt width;
        UInt height;
        Byte * img;
} ByteImage;
*/
```

Visual Basic:

```
Public Type ByteImage
        width As Long ' UInt
        height As Long ' UInt
        img As Variant ' byte*
End Type
```

Field of structure	Type	Description
Width	`UInt`	Width of image in pixels.
Height	`UInt`	Height of image in pixels.
Img	`Byte*`	Pointer to the first element of array of bytes of size width*height which represents an image. 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 face image pixel (grayscale value). Value of `0` means black and value of `255` means white.

Only this representation of image is used through VeriLook library. Some handy functions to operate with this structure are already in the library. They are declared in the header `Sys.h`.

In Visual Basic and Access images are stored in arrays with lower bound `0` and upper bound `width*height-1`, all elements are byte values from `0` to `255` (from black to white) representing 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.

5.7.1. AllocByteImage function

Allocates memory for `ByteImage` representation of image.

C++:

```
ByteImage AllocByteImage(UInt width, UInt height);
```

Parameters:

[in]	width	Width of image
[in]	height	Height of image

Return values: `ByteImage` structure filled with indicated width, height and pointer to allocated memory for image.

5.7.2. AssignByteImage function

Assigns the image of VeriLook image representation style to `ByteImage` structure.

C++:

```
ByteImage AssignByteImage(Byte * img, UInt width, UInt height);
```

Parameters:

[in]	Img	Pointer to an array representing the image in VeriLook image representation style
[in]	width	Width of the image
[in]	Height	Height of the image

Return values: `ByteImage` structure filled with indicated width, height and image pointer.

5.7.3. FreeByteImage function

Frees memory allocated for `ByteImage` representation of image.

C++:

```
void FreeByteImage(ByteImage image);
```

Parameters:

[in]	image	Filled `ByteImage` structure

Return values: There are no return values.

5.7.4. FillByteImage function

Fills `ByteImage` with indicated value.

C++:

```
void FillByteImage(ByteImage image, Byte value);
```

Parameters:

[in]	image	`ByteImage` that is supposed to be filled
[in]	value	The image will be filled with this value

Return values: There are no return values.

5.7.5. CopyByteImage function

Copies source `ByteImage` to the target, reallocates target's memory if necessary.

C++:

```
ByteImage CopyByteImage(ByteImage trg, ByteImage src);
```

Parameters:

[in]	trg	Destination image(trg.img must be valid pointer to the amount of memory described by trg.width and trg.height or `NULL`)
[in]	src	Source image

Return values: `ByteImage` structure using same memory as trg argument but containing a copy of information from src argument.

Example:

C++:

```
Byte * img = new Byte[640 * 480];
// allocate new image
ByteImage image1 = AllocByteImage(640, 480);
// assign already allocated image to ByteImage structure
ByteImage image2 = AssignByteImage(img, 640, 480);
// fill the first image with value 128
FillByteImage(image1, 128);
```

```
// copy information from the first image to the second
image2 = CopyByteImage(image2, image1);
// free allocated memory
FreeByteImage(image1);
FreeByteImage(image2);
// note that array "img" cannot be freed because it was assigned
// to image2 and was already freed by the above sentence
```

5.8. Face detection

Face detection is used to locate precise position of the face in the image identified by rectangle of face or coordinates of left and right eyes. The rectangle or/and eyes' coordinates can be localized automatically using the functions from following sections.

`VL_FRAME_DETAILS` or `VL_FRAME_DETAILS_EX` structure is used in single face localization functions to retrieve the details of face detection process. The common field of structures size must be filled with the size of structure to use them in single face localization functions.

Field of structure	Type	Description
`Size`	Int	Size of structure – `sizeof(VL_FRAME_DETAILS)` or `sizeof(VL_FRAME_DETAILS_EX)`
`foundFace`	Bool	Is set to `TRUE` if face is found, and `FALSE` – otherwise.
`leftEye`	Point	Is set to located coordinate of left eye if face is found
`rightEye`	Point	Is set to located coordinate of right eye if face is found
`similarity`	Double	Is set to similarity of detected object to face. Higher similarity – higher probability to be a face.
`symmetry`	Double	Is set to similarity of detected object to frontal face. Lower symmetry – higher probability to be a face that is suitable for face verification.
`eyeCandidates`	Int	Result of refinement of eyes coordinates. If refinement was successful `eyeCandidates` is set to the value that is greater or equal to zero. Otherwise it is set to the value that is lower than zero.

Example:

C++:

```
/* these structures are already defined in "Vlook.Interface.h"
struct VL_FRAME_DETAILS {
        Int size;
        Bool foundFace;
        Point leftEye;
        Point rightEye;
};
struct VL_FRAME_DETAILS_EX {
        Int size;
        Bool foundFace;
        Point leftEye;
        Point rightEye;
        Double similarity;
        Double symmetry;
        Int eyeCandidates;
};
*/

{
VL_FRAME_DETAILS details;
details.size = sizeof(VL_FRAME_DETAILS);
/* use structure in face detection functions */
VL_FRAME_DETAILS_EX details_ex;
details_ex.size = sizeof(VL_FRAME_DETAILS_EX);
/* use structure in face detection functions. Do not forget to typecast
VL_FRAME_DETAILS_EX structure to VL_FRAME_DETAILS when using it
as an argument for face detection functions */
}
```

Visual Basic:

```
Public Type VL_FRAME_DETAILS
        size As Long ' Int
        foundFace As Boolean ' Bool; true, if face is found
        leftEye As Point ' left eye, if face is found
        rightEye As Point ' right eye, if face is found
End Type

Public Type VL_FRAME_DETAILS_EX
        size As Long ' Int
        foundFace As Boolean ' Bool; true, if face is found
        leftEye As Point ' left eye, if face is found
        rightEye As Point ' right eye, if face is found
        similarity As Double ' similarity
        symmetry As Double ' symmetry
        eyeCandidates As Long ' used in frame processor only
        lastFaceBefore As Long ' Int; amount of frames from last seen face
End Type

' …
Dim details as VL_FRAME_DETAILS
details.size = VL_FRAME_DETAILS_SIZE
' use structure in face detection functions
Dim details_ex as VL_FRAME_DETAILS_EX
details_ex.size = VL_FRAME_DETAILS_EX_SIZE
' use structure in face detection functions. Do not forget to typecast
' VL_FRAME_DETAILS_EX structure to VL_FRAME_DETAILS when using it
' as an argument for face detection functions
```

C#:

```
public struct VLFrameDetails
{
        public int size ;

        public bool foundFace;
        public Point leftEye;
        public Point rightEye;
}

public struct VLFrameDetailsEx
{
        public int size ;

        public bool foundFace;
        public Point leftEye;
        public Point rightEye;
        public double similarity;
        public double symmetry;
        public int eyeCandidates;
}


VeriLook VLN = new VeriLook();
VeriLook.VLFrameDetailsEx detailsEx =
                                new VeriLookT.VLFrameDetailsEx();
detailsEx.size = Marshal.SizeOf(detailsEx);
VLN.DetectFaceOnce(imageArray, width, height, ref detailsEx);

VeriLook VLN = new VeriLook();
VeriLook.VLFrameDetails details = new VeriLook.VLFrameDetails();
details.size = Marshal.SizeOf(details);
VLN.DetectFaceOnce(imageArray, width, height, ref details);
```

VL_FACE structure is used in localization of multiple faces functions to retrieve the details of face detection process. First of all rectangles with faces are found in the image. It depends on VeriLook specific parameter DetectFace.detectEyesOnFaces how many face rectangles will be used to detect precise positions of eyes on them. No special initialization for this structure is needed.

Field of structure	Type	Description
topLeft	Point	Is set to top left corner of face rectangle.
bottomRight	Point	Is set to bottom right corner of face rectangle.
similarity	Double	Is set to similarity of detected object to face. Higher similarity – higher probability to be a face.
leftEye	Point	Is set to located coordinate of left eye if asked according to VeriLook specific parameter DetectFace.detectEyesOnFaces.
rightEye	Point	Is set to located coordinate of right eye if asked according to VeriLook specific parameter DetectFace.detectEyesOnFaces.
eyePrecision	Int	Result of refinement of eyes coordinates. If refinement was successful eyeCandidates is set to the value that is greater or equal to zero. Otherwise it is set to the value that is lower than zero. Zero means the best possible refinement, higher values – lower quality of refinement.

5.8.1. VLDetectFaceOnce function

The function is supposed to be used for detecting faces in static images.

It finds whether any face is present in the image. If the image was identified as containing human face that passes face verification requirements, coordinates of its left and right eyes are located.

The function uses features extraction and VeriLook specific parameters.

C++:

```
Int VLOOK_API VLDetectFaceOnce(
        ByteImage image, VL_FRAME_DETAILS* details, void* context);
```

Visual Basic:

```
Public Declare Function VLDetectFaceOnce _
        Lib "VLVBP.DLL" Alias "VBVLDetectFaceOnce" _
        (ByRef image As ByteImage, ByRef details As Any, _
         ByVal context As Long) As Long
```

C#:

```
public void DetectFaceOnce(
        byte[] image,
        uint width,
        uint height,
        ref VLFrameDetails details
);

public void DetectFaceOnce(
        byte[] image,
        uint width,
        uint height,
        ref VLFrameDetailsEx detailsEx
);
```

Parameters:

[in]	image	An image
[out]	details	A structure containing details of detection process
	context	Context to perform features extraction in. Null for default context

Return values: If VeriLook library is not registered returns `VLE_NOT_REGISTERED`.If context is `null` and VeriLook library is not initialized returns `VLE_NOT_INITIALIZED`.If face image width or height is not in legal range returns `VLE_ILLEGAL_IMAGE_SIZE`.If image is `null` returns `VLE_ARGUMENT_NULL`.Otherwise performs face detection in static image and returns either `VLE_OK` or `VLE_LOW_QUALITY_IMAGE` (if face image quality is low). `VLE_LOW_QUALITY_IMAGE` is only a warning. Calling application can either ignore it or use another image.

Example:

C++:

```
// Face detection in static images function
ByteImage image = /* obtain face image */;
```

```
VL_FRAME_DETAILS details;
details.size = sizeof(VL_FRAME_DETAILS);
if (VLSucceeded(VLDetectFaceOnce(image, &details, NULL)))
{
        if (details.foundFace)
        {
                /* use "details.leftEye" and "details.rightEye"
                to visualize results of face localization */
        }
}
```

Visual Basic:

```
' Face detection in static images function
Dim image as ByteImage
image = ' obtain face image
Dim details as VL_FRAME_DETAILS
details.size = VL_FRAME_DETAILS_SIZE
If VLSucceeded(VLDetectFaceOnce(image, details, VL_DEFAULT_CONTEXT)) Then
        If details.foundFace Then
                ' use "details.leftEye" and "details.rightEye"
                ' to visualize results of face localization
        End If
End If
```

C#:

```
// Face detection in static images function
VeriLook VLN = new VeriLook();
try
{
        VeriLook.VLDetailsEx detailsEx =
                        new VeriLook.VLDetailsEx();
        details_ex.size = Marshal.SizeOf(detailsEx);
        VLN.DetectFaceOnce(imageArray, width, height, ref detailsEx);
        if(detailsEx.foundFace)
        {
                /* use "details.leftEye" and "details.rightEye"
                to visualize results of face localization */
        }
}
catch(Exception ex)
{
        MessageBox.Show(ex.Message);
}
```

5.8.2. VLDetectMultipleFacesOnce function

The function is supposed to be used for detecting multiple faces in static images.

It finds whether any face is present in the image. If any human face that passes face verification requirements are found, rectangles of faces are located. It depends on VeriLook specific parameter DetectFace.detectEyesOnFaces whether coordinates of their left and right eyes are located.

The function uses VeriLook specific parameters.

C++:

```
Int VLOOK_API VLDetectMultipleFacesOnce(
        ByteImage image, VL_FACE * faces, Int * facesCount, void * context);
```

[Visual Basic]

```
Public Declare Function VLDetectMultipleFacesOnce _
        Lib "VLVBP.DLL" Alias "VBVLDetectMultipleFacesOnce" _
        (ByRef image As ByteImage, faces As VL_FACES, _
         ByRef facesCount As Long, ByVal context As Long) As Long
```

C#:

```
public VLFace[] DetectMultipleFacesOnce(
        byte[] image,
        uint width,
        uint height,
        out int facesCount
);
```

Parameters:

[in]	image	An image
[out]	faces	Array of VL_FACE structures containing details of detection process
[out]	facesCount	Number of VL_FACE structures filled, same as number of detected faces
	context	Context to perform features extraction in. Null for default context

Return values: If VeriLook library is not registered returns `VLE_NOT_REGISTERED`.If context is `null` and VeriLook library is not initialized returns `VLE_NOT_INITIALIZED`.If face image width or height is not in legal range returns `VLE_ILLEGAL_IMAGE_SIZE`.If image is `null` returns `VLE_ARGUMENT_NULL`.Otherwise performs face detection in static image and returns either `VLE_OK` or `VLE_LOW_QUALITY_IMAGE` (if face image quality is low). `VLE_LOW_QUALITY_IMAGE` is only a warning. Calling application can either ignore it or use another image.

Example:

C++:

```
// Face detection in static images function
ByteImage image = /* obtain face image */;
Int maxFaces, facesCount;
VLGetParameter("DetectFace.maxFaces", (void *) &maxFaces, NULL);
VL_FACE * faces = new VL_FACE[maxFaces];
if (VLSucceeded(VLDetectMultipleFacesOnce(image, faces, &facesCount, NULL)))
{
        if (facesCount > 0)
        {
                /* use details in VL_FACE structures
                to visualize results of face localization */
        }
}
delete[] faces;
```

Visual Basic:

```
' Face detection in static images function
Dim image as ByteImage
image = ' obtain face image
Dim faces as VL_FACES
If VLSucceeded(VLDetectMultipleFacesOnce( _
        image, faces, facesCount, VL_DEFAULT_CONTEXT)) Then
                If facesCount > 0 Then
                        ' use details in VL_FACE structures
                        ' to visualize results of face localization
                        End If
End If
```

C#:

```
// Face detection in static images function
byte[] imageArray = /* obtain face image */;
int facesCount;
VeriLook.VLFace[] VLFace;
int facesCount;
VLFace = VLN.DetectMultipleFacesOnce(
                imageArray, width, height, out facesCount);
```

```
if (facesCount > 0)
{
/* use details in VLFace structures
to visualize results of face localization */
}
```

5.8.3. VLDetectFace function

This function is supposed to be used for detecting faces in a sequence of images. Every image from the sequence (later referred as frame) must be processed by the function one by one until the function indicates that no more frames can help to find the better representation of current face for extracting features.

Details of possible face location (identified by coordinates of the eyes) are returned after processing every frame. Once the function indicates the end of face detection in the sequence of images, the best representation of current face is returned along with the details of face location.

The function uses features extraction and VeriLook specific parameters.

C++:

```
Int VLOOK_API VLDetectFace(
        ByteImage image, VL_FRAME_DETAILS* details, void* context);
```

Visual Basic:

```
Public Declare Function VLDetectFace _
        Lib "VLVBP.DLL" Alias "VBVLDetectFace" _
        (ByRef image As ByteImage, ByRef details As Any, _
         ByVal context As Long) As Long
```

C#:

```
public int DetectFace(
        byte[] image,
        uint width,
        uint height,
        ref VLFrameDetails details
);

public int DetectFace(
        byte[] image,
        uint width,
        uint height,
        ref VLFrameDetailsEx details
);
```

Parameters:

[in]	image	An image
[out]	details	A structure containing details of detection process
	context	Context to perform features extraction in. Null for default context

Return values: If VeriLook library is not registered returns `VLE_NOT_REGISTERED`.If context is `null` and VeriLook library is not initialized returns `VLE_NOT_INITIALIZED`.If face image width or height is not in legal range returns `VLE_ILLEGAL_IMAGE_SIZE`.If image is `null` returns `VLE_ARGUMENT_NULL`.Otherwise performs face detection in current frame using the information from earlier processed frames (if any) and returns either `VLE_OK` (if face detection in the sequence of images is finished) or `VLE_FALSE` (if more frames are needed for localization of face).

Example:

C++:

```
// Face detection in sequence of images function
ByteImage image = /* obtain frame from the sequence */;
VL_FRAME_DETAILS details;
details.size = sizeof(VL_FRAME_DETAILS);
Int res = VLDetectFace(image, &details, NULL);
if (VLSucceeded(res))
{
        if (res == VLE_FALSE)
        {
                if (details.foundFace)
                {
                        /* "details" are the details of face localization
                        in current frame */
                        /* use "details.leftEye" and "details.rightEye"
                        to visualize results of face localization */
                }
        }
        else if (res == VLE_OK)
        {
                if (details.foundFace)
                {
                        /* these are the details of face localization
                         in all sequence of images */
                        /* "image" now contains the most suitable image
                         for face detection of current face */
                        /* use "details.leftEye" and "details.rightEye"
                         to visualize results of face localization */
                }
        }
}
```

Visual Basic:

```
' Function for face detection in image sequence

Dim image as ByteImage
image = ' obtain frame from the sequence
Dim details as VL_FRAME_DETAILS
details.size = VL_FRAME_DETAILS_SIZE
Dim res as Long
res = VLDetectFace(image, details, VL_DEFAULT_CONTEXT)
If VLSucceeded(res) Then
        If res = VLE_FALSE Then
                If details.foundFace Then
                ' "details" are the details of face localization
                ' in current frame
                ' use "details.leftEye" and "details.rightEye"
                ' to visualize results of face localization
                End If
        Else
                If res = VLE_OK Then
                        If details.foundFace Then
                        ' these are the details of face localization
                        ' in all sequence of images
                        ' "image" now contains the most suitable image
                        ' for face detection of current face
                        ' use "details.leftEye" and "details.rightEye"
                        ' to visualize results of face localization
                        End If
                End If
        End If
End If
```

C#:

```
// Face detection in sequence of images function
VeriLook VLN = new VeriLook();
/* obtain frame from the sequence into imageArray*/
VeriLook.VLFrameDetails detailsEx = new VeriLook.VLFrameDetails();
detailsEx.size = Marshal.SizeOf(VLFrame);
int ret = VLN.DetectFace(imageArray, width, height, ref detailsEx);
if (res == VeriLookException.False)
{
        if (details.foundFace)
        {
                /* "details" are the details of face localization
                in current frame */
                /* use "details.leftEye" and "details.rightEye"
                to visualize results of face localization */
        }
}
else if (res == VeriLookException.Ok)
{
        if (details.foundFace)
        {
                /* these are the details of face localization
                in all sequence of images */
                /* "image" now contains the most suitable image
                for face detection of current face */
                /* use "details.leftEye" and "details.rightEye"
                to visualize results of face localization */
        }
}
```

5.9. Features extraction

You can use features extraction to extract feature template from face image and then store it in a database (enroll face). For more information see Face images and Features.

Use VLExtract function to perform features extraction.

5.9.1. VLExtract function

Extracts features from an image containing face indicated by the coordinates of left and right eyes. These coordinates can be registered manually or located automatically using VLDetect-FaceOnce or VLDetectFace functions.

Features are extracted to an array of bytes. Feature template size can be retrieved from the context passing the `"featuresSize"` string as an argument for VLGetParameter function. Memory for the features array must be allocated before calling function VLExtract.

It is recommended to save all enrolled images to allow re-enrolling in case of changes in internal feature template extraction algorithm in upcoming versions of VeriLook SDK.

The function uses features extraction and VeriLook specific parameters.

C++:

```
Int VLOOK_API VLExtract(
        ByteImage image, Point leftEye, Point rightEye, Byte* features,
        void* context);
```

Visual Basic:

```
Public Declare Function VLExtract _
        Lib "VLVBP.DLL" Alias "VBVLExtract" _
        (ByRef image As ByteImage, _
         ByRef leftEye As Point, ByRef rightEye As Point, _
         ByRef features As Variant, ByVal context As Long) As Long
```

C#:

```
public byte[] Extract(
        byte[] image,
        uint width,
        uint height,
        Point leftEye,
        Point rightEye
);
```

Parameters:

[in]	image	An image containing human face
[in]	leftEye	Coordinate of left eye.
[in]	rightEye	Coordinate of right eye.
[out]	features	Extracted features
	context	Context to perform features extraction in. Null for default context

Return values: If VeriLook library is not registered returns `VLE_NOT_REGISTERED`.If context is `null` and VeriLook library is not initialized returns `VLE_NOT_INITIALIZED`.If image or features is `null` returns `VLE_ARGUMENT_NULL`.Otherwise performs features extraction and returns either `VLE_OK` or `VLE_LOW_QUALITY_IMAGE` (if face image quality is low). `VLE_LOW_QUALITY_IMAGE` is only a warning. Calling application can either ignore it or get another image containing the face.

Example:

C++:

```
// Extraction function
DWord featuresSize;
```

```
VLGetParameter("featuresSize", &featuresSize, NULL);

ByteImage image;
Point leftEye, rightEye;
// 1) load an image from file or get it from camera
// 2) detect the positions of left and right eyes
Byte * features = new Byte[featuresSize];
VLExtract(image, leftEye, rightEye, features, NULL);
// 3) save features to use them for matching later
delete [] features;
```

Visual Basic:

```
' Extraction function

Dim featuresSize as Variant
VLGetParameter "featuresSize", featuresSize, VL_DEFAULT_CONTEXT
Dim image as ByteImage
Dim leftEye as Point
Dim rightEye as Point
' 1) load an image from file or get it from camera
' 2) detect the positions of left and right eyes
Dim features() as Byte
ReDim features(featuresSize)
VLExtract image, leftEye, rightEye, features, VL_DEFAULT_CONTEXT
' 3) save features to use them for matching later
```

C#:

```
// Extraction function
byte[] imageArray;
Point leftEye, rightEye;
// 1) load an image from file or get it from camera
// 2) detect the positions of left and right eyes
byte[] features = VLN.Extract(imageArray, width, height, leftEye, rightEye);
// 3) save features to use them for matching later
```

5.10. Features generalization

You can use features generalization to increase quality of the recognition. Generalization performs conjunction of several features collections to one collection. You can use features generalization during enrollment. To obtain features for generalization use features extraction functions.

Generalization uses similarity threshold for matching to determine if provided features collections are of the same person. For more information see Matching threshold.

Use VLGeneralize function to perform features generalization.

5.10.1. VLGeneralize function

Performs generalization of features collections in the specified context.

This function uses features extraction, features generalization, features matching and VeriLook specific parameters.

C++:

```
Int VLOOK_API VLGeneralize(
        Byte** featuresSets, Int setsCount, Byte* features,
        Double simThreshold, void* context);
```

Visual Basic:

```
Public Declare Function VLGeneralize _
        Lib "VLVBP.DLL" Alias "VBVLGeneralize" _
        (ByRef featuresSets As Variant, ByVal setsCount As Long, _
         ByRef features As Variant, ByVal simThreshold As Double, _
         ByVal context As Long) As Long
```

C#:

```
public byte[] Generalize(
        byte[][] featuresSets,
        int setsCount,
        double simThreshold
);
```

Parameters:

[in]	featuresSets	Array of features collections to generalize
[in]	setsCount	Number of features collections.
[out]	features	After execution of the function contains generalized features
[in]	simThreshold	Similarity threshold for matching templates being generalized
	Context	Context to perform features generalization in. Null for default context

Return values: If VeriLook library is not registered returns `VLE_NOT_REGISTERED`.If context is `null` and VeriLook library is not initialized returns `VLE_NOT_INITIALIZED`.If featuresSets is `null` returns `VLE_ARGUMENT_NULL`.If features collections cannot be generalized returns `VLE_FAILED`.Otherwise returns `VLE_OK`.

Example:

C++:

```
// Generalization function
Byte *feats[3];
Byte features[VL_MAX_FEATURES_SIZE];
Dword size;
Float simThreshold = … // Obtain generalization threshold

feats[0] = /* obtain first face features */;
feats[1] = /* obtain second face features */;
feats[2] = /* obtain third face features */;
if (VLSucceeded(VLGeneralize(3, feats, features, &size, simThreshold, NULL))
        printf("Generalization succeeded");
else
        printf("Generalization failed");
```

Visual Basic:

```
' Generalization function

Dim Feats(2) as Variant ' will hold 3 arrays of face features
Dim Features(VL_MAX_FEATURES_SIZE) as Byte
Dim Size as Long

Dim simThreshold as Double
simThreshold = … ' Obtain generalization threshold

Feats(0) = ' obtain first face features
Feats(1) = ' obtain second face features
Feats(2) = ' obtain third face features
If VLSucceeded(VLGeneralize( _
        3, Feats, Features, Size, simThreshold, VL_DEFAULT_CONTEXT)) Then
                MsgBox "Generalization succeeded"
Else
        MsgBox "Generalization failed"
End if
```

C#:

```
// Generalization function
VeriLook VLN = new VeriLook();
```

```
byte[][] featuresSets;
featuresSets ... /* obtain face features */;
byte[] features;
double simThreshold = ...// Obtain generalization threshold
try
{
        features = VLN.Generalize(featuresSets, setsCount, simThreshold);
        MessageBox.Show("Generalization succeeded");
}
catch(VeriLookException ex)
{
        if(ex.ErrorCode == VeriLookException.Failed)
        {
                MessageBox.Show("Generalization failed");
        }
}
```

5.11. Verification

You can use verification to determine if two features collections are of the same subject. It uses `VLP_MATCHING_THRESHOLD` parameter. To obtain features from face image use features extraction functions. Also you may use features generalization functions to increase recognition reliability.

Use VLVerify function to perform verification.

5.11.1. VLVerify function

Calculates the probability of match between two features collections in the specified context.

This function uses features matching and VeriLook specific parameters. For more information see Parameters.

C++:

```
Int VLOOK_API VLVerify(
        Byte* features1, Byte* features2, Double* similarity, void* context);
```

Visual Basic:

```
Public Declare Function VLVerify _
        Lib "VLVBP.DLL" Alias "VBVLVerify" _
        (features1 As Variant, features2 As Variant, _
         ByRef similarity As Double, _
         ByVal context As Long) As Long
```

C#:

```
public double Verify(byte[] features1, byte[] features2);
```

Parameters:

[in]	features1	First face features
[in]	features2	Second face features
[out]	similarity	Similarity between first and second face.
	context	Context to perform verification in. Null for default context

Return values: If VeriLook library is not registered returns VLE_NOT_REGISTERED.If context is null and VeriLook library is not initialized returns VLE_NOT_INITIALIZED.If identification is started returns VLE_INVALID_MODE.If one of the features collections is null returns VLE_ARGUMENT_NULL.If insufficient memory then returns VLE_OUT_OF_MEMORY.Otherwise returns VLE_OK.

Example:

C++:

```
// Verification function
Byte *features1, *features2;
Bool ok = FALSE;
Double similarity = 0.0;
features1 = /* obtain first face features */;
features2 = /* obtain second face features */;
ok = VLSucceeded(VLVerify(features1, features2, &similarity, NULL));
Double verifyThreshold = /* obtain similarity threshold for verification */
if (ok)
{
        if (similarity > verifyThreshold)
                printf("Same subject. Similarity: %d", similarity);
        else
                printf("Different subjects. Similarity: %d", similarity);
}
```

Visual Basic:

```
' Verification function
Dim features1() as Byte
Dim features2() as Byte
Dim ok as Boolean
```

```
ok = FALSE
Dim similarity as Double
similarity = 0.0
features1 = ' obtain first face features
features2 = ' obtain second face features
ok = VLSucceeded(VLVerify( _
        features1, features2, &similarity, VL_DEFAULT_CONTEXT))
Dim verifyThreshold as Double
verifyThreshold = ' obtain similarity threshold for verification
If ok Then
        If similarity > verifyThreshold then
                MessageBox("Same subject. Similarity: " & CStr(similarity))
        Else
                MessageBox("Different subjects. Similarity: "& similarity))
        End If
End If
```

C#:

```
// Verification function
byte[] features1, features;
double similarity = 0.0;
features1 = /* obtain first face features */;
features2 = /* obtain second face features */;
similarity = VLN.Verify(features, featuresFromDB);
double verifyThreshold = /* obtain similarity threshold for verification */
if (similarity > verifyThreshold)
{
        MessageBox.Show(string.Format("Same subject. Similarity: {0}",
                                                           similarity));
}
else
{
        printf(string.Format("Different subjects. Similarity: {0}",
                                                        similarity));
}
```

5.11.2. Matching threshold and similarity

VeriLook features matching algorithm provides value of features collections similarity as a result. The higher is similarity, the higher is probability that features collections are obtained from the same person.

Matching threshold is linked to false acceptance rate (FAR, different subjects erroneously accepted as of the same) of matching algorithm. The higher is threshold, the lower is FAR and higher FRR (false rejection rate, same subjects erroneously accepted as different) and vice a versa.

FAR	Threshold
1%	0.520
0.1%	0.585
0.01%	0.642
0.001%	0.695
0.0001%	0.760

In order to improve person identification one can implement multiple matching attempts. For details refer to the demo program source code and documentation.

5.12. Face features

Features or features collection or template are data extracted from face image that is used in verification. To obtain features from face image use features extraction functions. You may also use features generalization to improve quality of the features.

Features are stored in array of bytes. Number of bytes occupied by features can be taken from the context passing the `"featuresSize"` string as an argument for VLGetParameter function.

Chapter 6. Sample applications

Sample applications are provided for Visual C++, Visual Basic and Access (VBA).

6.1. Visual C++

Source of Visual C++ application is located in `VLDemo.cpp\` subdirectory of SDK (if you have VeriLook library as .lib then the application is located in `VLDemo.lib\` subdirectory). Project file for C is `VLDemo.cpp.vcproj` (version 7.0) or `VLDemo_cpp.dsp` (version 6.0) (`VLDemo.lib.csproj` and `VLDemo_lib.dsp` for .lib version of VeriLook). Both applications share resource files that are located in `Res\` subdirectory of SDK:

`Logo.bmp`	Logo picture in bitmap format
`VLDemo.ico`	Demo application icon
`VLDemoSmall.ico`	Small application icon
`VLDemo.exe.manifest`	Manifest file
`VLDemo.mdb`	Default database

Interface for VeriLook library is provided in header file `VLook.Interface.h` in `Include\` subdirectory of SDK. C application also uses `VLook.lib` library file in `Lib\` subdirectory of SDK. For .lib version of VeriLook library you have to include `VLook.lib` file from `VLLib\` subdirectory into your C application project. C sample application uses MFC as a GUI framework. Application utilizes `CSImage` class for image processing, thin wrapper of Microsoft GDI+ `CImage` class.

`MainFrm.h` and `MainFrm.cpp`	Main application logic is implemented here.
`VLDemoDoc.h` and `VLDemoDoc.cpp`	Document part of Document-View interface.
`OptionsDlg.h` and `OptionsDlg.cpp`	Application options editing dialog (`COptionsDlg`).
`RegistrationDlg.h` and `RegistrationDlg.cpp`	VeriLook library registration dialog (`CRegistrationDlg`).
`VLDemoView.h` and `VLDemoView.cpp`	View part of Document-View interface for displaying enrolled and matched images.
`FaceView.h` and `FaceView.cpp`	View part of Document-View interface for displaying frames from video capture device.
`EnrollDlg.h` and `En-`	Dialog for entering subject identifier

rollDlg.cpp	
VLDemo.h and VLDemo.cpp	Main application.
LogView.h and LogView.cpp	CListView descendant for application message logging.
StdLogView.h and StdLogView.cpp	CView descendant for message logging
Capturer.h and Capturer.cpp	Helper class for video capture device enumeration and video capturing.
DetectionData.h and DetectionData.cpp	CDetectionData class encapsulating data received from face detection algorithm.
FaceDatabase.h and FaceDatabase.cpp	CFaceDatabase class for enrolling and matching face templates.
SampleGrabber.h and SampleGrabber.cpp	ISampleGrabber interface implementation utilized by CCapturer for sending frames from video capture device to application.
SImage.h and SImage.cpp	CSImage class for loading, saving and displaying images. Thin wrapper of Microsoft GDI+ CImage.
SplitterWndEx.h and SplitterWndEx.cpp	CSplitterWnd descendant with ability to set split pane ratios.
SProfiler.h and SProfiler.cpp	Simple class for timing application execution. Thin wrapper of WinAPI function QueryPerformanceFrequency and QueryPerformanceCounter
Utils.h and Utils.h	Misc. utility functions.
utilsSafearray.h and utilsSafearray.h	Functions dealing with SAFEARRAY.
VLDatabase.h and VLDatabase.cpp	Face feature storage and retrieval from ADO database.
VLSettings.h and VLSettings.cpp	Application settings storage class.
IStdOut.h and IStdOut.cpp	Application logging interface.
Mtype.h, FourCC.h, getdxver.cpp	Utility files from Microsoft DirectX SDK

When main form window is created (CMainFrame::OnCreate method) Source menu is filled with video capture devices enumerated using Microsoft DirectX (`CMainFrame::FillDevicesMenu` in C), also Source»File menu item is appended to allow enrolling or matching static images. If there was at least one camera detected, application starts displaying live video, otherwise Source»File is selected.

When application is initializing (`CVLDemoApp::InitInstance`) registration dialog is shown if VeriLook library is not registered, system and VeriLook library information is displayed in the log and face database is loaded. On main application form creation (`CMainFrame::OnCreate`) "Source" menu is filled with video capture devices and first menu item is activated.

Menu commands are described in following table:

Menu command	Description
Source»Camera	Choose selected camera as video source
Source»File	Select an image file as a source.
Jobs»Enroll	Enroll image to face database.
Jobs»Enroll with generalization	Enroll several generalized images to face database.
Jobs»Match	Search for matching image in face database.
Tools»Face detection preview	View face detection result overlaid on images.
Tools»Save image	Save image to disk.
Tools»Clear logs	Clear application log windows.
Tools»Empty database	Empty face database.
Tools»Options…	Display options dialog.
Help»About VeriLook…	Display information about VeriLook demo application.

6.2. Visual C#

Source of Visual C# VeriLook demo application is located in `VLDemo.NET\VLDemo.net.cs` subdirectory of SDK.

To simplify work with DirectX special ActiveX was developed – VeriLook Webcam Capturer ActiveX (`VLCapturer.dll`). `VLCapturer.dll` **has to be registered using `bin\VLCapturerReg.bat` before running this sample. It is necessary in working direktory to have files: `VLook.dll`, `Neurotec.Biometrics.VeriLook.dll`, `VLCapturer.dll`, `AxInterop.VLCAPTURERLib.dll`, `Interop.VLCAPTURERLib.dll`, `VLDemo.mdb`.** Source is located in VLDemo.bas\VLCapturer\ subdirectory.

Visual C# sample application directory contents is listed and described here:

VLDemo.NET\VLDemo.net.cs	
Project File	
File	Description
`VL-Demo.net.cs.csproj`	Visual C# project file.
Forms	
File	Description
`MainForm.cs`	Main application logic is implemented here.
`MainForm.resx`	Resource information associated with `MainWindow.cs`.
`AboutForm.cs`	Application information form.
`AboutForm.resx`	Resource information associated with `AboutForm.cs`.
`ConfirmForm.cs`	Confirmation Form for clearing DataBase.
`ConfirmForm.resx`	Resource information associated with `ConfrimForm.cs`.
`DeviceForm.cs`	Form for showing video devices.
`DeviceForm.resx`	Resource information associated with `DeviceForm.cs`.
`EnrollForm.cs`	Form for entering image id.
`EnrollForm.resx`	Resource information associated with `EnrollForm.cs`.
`FaceCollection.cs`	Loads database into ArrayList.
`VideoControl.cs`	Control for showing images.
`RegisterForm.cs`	VeriLook DLL registration Form.
`RegisterForm.resx`	Resource information associated with `RegisterForm.cs`.
`SettingsFrom.cs`	Application settings Form. Allows changing of various application settings.
`SettingsFrom.resx`	Resource information associated with `SettingsFrom.cs`.
`Match.cs`	Dedicated for matching features.
`Demo.ico`	Demo application icon.
`AssemblyInfo.cs`	File contains assembly attributes.

Database	
File	Description
`VLDemo.mdb`	Access database.
XML	
File	Description
`VL-Demo.net.register.xml`	Registration information.
`VL-Demo.net.settings.xml`	Parameters information.

Menu commands are described in following table:

Menu command	Description
File»File	Select an image file as a source.
File»Device	Select a camera as a source.
File»Exit	Close VeriLook demo.
Jobs»Enroll	Enroll image to face database.
Jobs»Enroll with generalization	Enroll several generalized images to face database.
Jobs»Match	Search for matching image in face database.
Tools»Face detection preview	View face detection result overlaid on images.
Tools»Clear log	Clear application log window.
Tools»Clear database	Empty face database.
Tools»Settings…	Display application settings dialog.
Help»Register DLL…	Display VeriLook DLL registration dialog.
Help»About VeriLook…	Display information about VeriLook demo application.

As it is impossible to use VeriLook DLL directly from Visual C# special wrapper was written. Wrapper source is located in `VLDemo.NET\Wrapper\` subdirectory.

6.2.1. VeriLook C# Wrapper

VeriLook C# wrapper functions do not return results code but **all functions throw exception (VeriLookException)** when error occurs. You can get error code using ErrorCode method.

Neurotec.Biometrics (Neurotec.Biometrics.VeriLook.dll wraps VLook.dll) namespace consist of two classes:

VeriLook, VeriLookException;

structures:

VLFace, VLFrameDetails, VLFrameDetailsEx;

enumeration RegistrationType.

VLDemo.NET\Wrapper	
Project File	
File	Description
`Neurotec.Biometrics.VeriLook.csproj`	Visual C# project file.
Wrapper	
File	Description
`VeriLookWrapper.cs`	Main wrapper for `VLook.dll`.
`VeriLookException.cs`	Exceptions and errors.
`AssemblyInfo.cs`	File contains assembly attributes.

Class VeriLook:

```
public class VeriLook : IDisposable
```

Function/Property	Description
`VeriLook`	`VeriLook()` Class VeriLook constructor initializes VeriLook library and creates the context.

`Dispose`	`Dispose()` Un-initializes VeriLook library, deletes the context.
Registration	
`RegistrationType`	`int RegistrationType{ get; }` For more information please see RegistrationType
`GenerateId`	`string GenerateId(string serial)` Returns registration id. For more information please see GenerateId
`Register`	`void Register(string serial, string key)` Register VeriLook library. If VeriLook is not protected, ErrorCode returns VLE_REGISTRATION_NOT_NEEDED. For more information please see Register.
Parameters	
`GetParameter`	`object GetParameter(string parameter)` Function throws exceptions: VeriLookException, ArgumentException. For more information please see GetParameter, C# example
`SetParameter`	`void SetParameter( string parameter, object parValue )` Function throws exceptions: VeriLookException, ArgumentException. For more information please see SetParameter
Face detection	
`DetectFaceOnce`	

	`void DetectFaceOnce(`
	`        byte[] image,`
	`        uint imWidth,`
	`        uint imHeight,`
	`        ref VLFrameDetails details`
	`)`
	
	`void DetectFaceOnce(`
	`        byte[] image,`
	`        uint imWidth,`
	`        uint imHeight,`
	`        ref VLFrameDetailsEx details_ex`
	`)`
	
	For more information please see DetectFaceOnce

`DetectMultipleFacesOnce`	`VLFace[] DetectMultipleFacesOnce(`
	`        byte[] image,`
	`        uint imWidth,`
	`        uint imHeight,`
	`        out int facesCount`
	`)`
	
	For more information please see DetectMultiple-FacesOnce

`DetectFace`	`int DetectFace(`
	`        byte[] image,`
	`        uint imWidth,`
	`        uint imHeight,`
	`        ref VLFrameDetails details`
	`)`
	
	`int DetectFace(`
	`        byte[] image,`
	`        uint imWidth,`
	`        uint imHeight,`
	`        ref VLFrameDetailsEx details`
	`)`
	
	For more information please see DetectFace

Features extraction	

`Extract`	`byte[] Extract(`
	`        byte[] image,`

	`        uint imWidth,`
	`        uint imHeight,`
	`        Point leftEye,`
	`        Point rightEye`
	`)`
	
	For more information please see Extract

Features generalization	

`Generalize`	`byte[] Generalize(`
	`        byte[][] setFeatures,`
	`        int setCount,`
	`        out byte[] featuresGeneralized,`
	`        double simThreshold`
	`)`
	
	For more information please see Generalize

Verification	

`Verify`	`double Verify(`
	`        byte[] features1,`
	`        byte[] features2`
	`)`
	
	Returns similarity between faces. For more information please see Verify

Constants of class `VeriLook`

```
public const string libraryName = "libraryName";
public const string mCopyright = "copyright";
public const string VersionHigh = "versionHigh";
public const string VersionLow = "versionLow";
public const string FeaturesSize = "featuresSize";

public const string DetectFaceMinIOD = "DetectFace.minIOD";
public const string DetectFaceMaxIOD = "DetectFace.maxIOD";
public const string DetectFaceMaxFaces = "DetectFace.maxFaces";
public const string DetectFaceDetectEyesOnFaces
                                = "DetectFace.detectEyesOnFaces";
public const string DetectFaceMaxFramesCount
                                = "DetectFace.maxFramesCount";
public const string DetectFaceSimilarityThreshold
```

```
                                = "DetectFace.similarityThreshold";
public const string DetectFaceSymmetryThreshold
                                = "DetectFace.symmetryThreshold";
```

For more information please see Parameters

Neurotec.Biometrics.VeriLook.dll structures:

```
public struct VLFace
{
        public Point topLeft;
        public Point bottomRight;
        public double similarity;
        public Point leftEye;
        public Point rightEye;
        public int eyePrecision;
}

public struct VLFrameDetails
{
        public int size ;

        public bool foundFace;
        public Point leftEye;
        public Point rightEye;
}

public struct VLFrameDetailsEx
{
        public int size ;

        public bool foundFace;
        public Point leftEye;
        public Point rightEye;
        public double similarity;
        public double symmetry;
        public int eyeCandidates;
}
```

For more information please see VL_FACE, VL_FRAME_DETAILS, VL_FRAME_DETAILS_EX

Neurotec.Biometrics.VeriLook.dll enumerations:

```
public enum RegistrationType
{
        NotProtected = 0,
        Hasp = 1,
        Pc = 2,
        Uareu = 4,
        Unregistered = 6,
        Lan = 8,
}
```

For more information please see VLRegistrationType.

Class `VeriLookException`:

```
public class VeriLookException : Exception
```

Constructor	Description
`VeriLookException`	`internal VeriLookException( string msg, int errorCode ):base(msg)`
`ErrorCode`	`public int ErrorCode { get; }` Gets error code.
`Message`	`public override string Message { get: }` Gets error code and description.

Constants of class `VeriLookException`:

```
public const int False = 1;
public const int Ok = 0;
public const int Failed = -1;
public const int OutOfMemory = -2;
public const int NotInitialized = -3;
public const int ArgumentNull = -4;
public const int InvalidArgument = -5;
public const int InvalidTemplate = -6;
public const int TemplatesNotCompatible = -7;
public const int NotImplemented = -9;

public const int InvalidParameter = -10;
public const int ParameterReadOnly = -11;

public const int NotRegistered = -2000;
public const int InvalidSerialNumber = -2001;
public const int InvalidRegistrationKey = -2002;
public const int ScannerDriverError = -2003;
public const int RegistrationNotNeeded = -2004;
public const int NoScanner = -2005;
public const int MoreThenOneScanner = -2006;
public const int LMConnectionError = -2007;
public const int LMNoMoreLicenses = -2008;

public const int IllegalImageResolution = -101;
public const int IllegalImageSize = -102;
public const int LowQualityImage = -103;

public const int InvalidMode = -1000;
```

For more information please see Error codes

6.3. Visual Basic

Source of Visual Basic VeriLook demo application is located in `VLDemo.bas\` subdirectory of SDK.

Visual Basic sample application directory contents is listed and described here:

Forms	
File	Description
`Main.frm`	Sample application Main form. Contains main menu, two picture boxes and log window.
`Main.frx`	Main form data (binary file; used by Visual Basic).
`DialogRegister.frm`	VeriLook DLL registration form.
`DialogRegister.frx`	Registration form data (binary file; used by Visual Basic).
`EnrollDialog.frm`	Enroll form.
`EnrollDialog.frx`	Enroll form data (binary file; used by Visual Basic).
`SettingForm.frm`	Application settings form. Allows changing of various application settings.
`SettingsForm.frx`	Settings form data (binary file; used by Visual Basic).

`frmAbout.frm`	Application information form.
`frmAbout.frx`	Information form data (binary file; used by Visual Basic).
Modules	
File	Description
`AppSettings.bas`	Application settings.
`DataBase.bas`	Operations with database.
`SaveInfo.bas`	Helper functions/sub for face enrollment.
`Service.bas`	Useful collections of functions/sub for face features drawing and image manipulations.
`VLook.bas`	VeriLook interface declaration and implementation of helper functions.
`Types.bas`	Miscellaneous types and functions/sub.
Class Modules	
File	Description
`FaceRecord.cls`	Class which defines face record.
Database	
File	Description
`VLDemo.mdb`	Access database.
DLLs	
File	Description
`VLook.dll`	VeriLook 2.0 DLL.
`VLVBP.dll`	VeriLook 2.0 Parser DLL.
Project Files	
File	Description
`VLookVBDemo.vbp`	Visual Basic Project.
`VLookVBDemo.vbw`	Visual Basic Project Work Space.

Menu commands are described in following table:

Menu command	Description
File»Exit	Close VeriLook demo.
Jobs»Enroll	Enroll image to face database.
Jobs»Enroll with generalization	Enroll several generalized images to face database.
Jobs»Match	Search for matching image in face database.
Tools»Face detection preview	View face detection result overlaid on images.
Tools»Clear log	Clear application log window.
Tools»Clear database	Empty face database.
Tools»Settings…	Display application settings dialog.
Help»Register DLL…	Display VeriLook DLL registration dialog.
Help»About VeriLook…	Display information about VeriLook demo application.

As it is impossible to use VeriLook DLL directly from Visual Basic special wrapper was written. Wrapper source is located in `VLDemo.bas\Parser\` subdirectory.

To simplify work with DirectX special ActiveX was developed – VeriLook Webcam Capturer ActiveX (`VLCapturer.dll`). Its source is located in `VLDemo.bas\VLCapturer\` subdirectory. `VLCapturer.dll` has to be registered using `bin\VLCapturerReg.bat` before running this sample. Source located in

6.3.1. VeriLook Visual Basic Wrapper

Wrapper files are listed here:

File	Description
`vlparser.sln`	Project solution file.
`vlparser.vcproj`	Project file.
`VLVBP.def`	Defines DLL exported functions.
`vlparser.cpp`	DLL functions implementation.
`VLVBP.rc`	DLL recourses.
`resource.h`	Defines recourses 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.
`ReadMe.txt`	ReadMe file

Parser allows Visual Basic applications to pass arrays or structures instead of pointers. There is a lot of code in parser source that manipulates SAFEARRAY data type. Please refer to MSDN (Microsoft Developer Network, http://msdn.microsoft.com/) for more information about SAFEARRAYs. Functions prefix `"VL"` was changed to `"VBVL"` to prevent name collision with VeriLook DLL functions.

Exported functions:

```
// Initialization
INT WINAPI VBVLInitialize();
INT WINAPI VBVLFinalize();
```

```
// Registration
INT WINAPI VBVLRegistrationType();
INT WINAPI VBVLGenerateId(CHAR * serial, CHAR * id);
INT WINAPI VBVLRegister(CHAR * serial, CHAR * key);
```

```
// Contexts
INT WINAPI VBVLCreateContext();
INT WINAPI VBVLFreeContext(INT context);
```

```
// Parameters
INT WINAPI VBVLGetParameter(CHAR* parameter, VARIANT* value, INT context);
INT WINAPI VBVLSetParameter(CHAR* parameter, VARIANT value, INT context);
```

```
// Face detection
Int WINAPI VBVLDetectFaceOnce(
        ByteImageVB* image, VL_FRAME_DETAILS * details, void * context);
Int WINAPI VBVLDetectMultipleFacesOnce(
        ByteImageVB* image, VL_FACE * faces, Int * facesCount,
        void * context);
Int WINAPI VBVLDetectFace(
        ByteImageVB* image, VL_FRAME_DETAILS * details, void * context);
```

```
// Features extraction
Int WINAPI VBVLExtract(
        ByteImageVB* image, Point* leftEye, Point* rightEye,
        VARIANT * VBfeatures, void * context);
```

```
// Verification
Int WINAPI VBVLVerify(
        VARIANT * VLEatures1, VARIANT * VLEatures2, Double * similarity,
        void * context);
```

```
// Features generalization
Int VLOOK_API VBVLGeneralize(
        VARIANT *vgen_features, Int count, VARIANT *VLEatures,
        Double simThreshold, void * 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 VBSaveImageToFile(
        CHAR* filename, INT width, INT height, VARIANT* vimage);
INT WINAPI VBDrawEyes(
        INT width, INT height, VARIANT* vimage, INT* handle,
        INT leyex, INT leyey, INT reyex, INT reyey);
```

6.3.2. Usage of VeriLook Visual Basic Parser

VeriLook Parser DLL exports following VeriLook functions wrappers:

Function	Description
`VBVLInitialize`	For more information please see VLInitialize
`VBVLFinalize`	For more information please see VLFinalize
`VBVLRegistrationType`	For more information please see VLRegistrationType
`VBVLGenerateId`	For more information please see VLGenerateId
`VBVLRegister`	For more information please see VLRegister
`VBVLCreateContext`	For more information please see VLCreateContext
`VBVLFreeContext`	For more information please see VLFreeContext
`VBVLGetParameter`	For more information please see VLGetParameter

`VBVLSetParameter`	For more information please see VLSetParameter
`VBVLExtract`	For more information please see VLExtract
`VBVLGeneralize`	For more information please see VLGeneralize
`VBVLVerify`	For more information please see VLVerify
`VBVLDetectFaceOnce`	For more information please see VLDetectFaceOnce
`VBVLDetectMultiple-FacesOnce`	For more information please see VLDetectMultiple-FacesOnce
`VBVLDetectFace`	For more information please see VLDetectFace

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

Function	Description
`VBImageToHandle`	Converts image array to handle. This function is useful when image must be passed to PictureBox control.
`VBHandleToImage`	Converts handle to image array.
`VBLoadImageFromFile`	Loads image from specified file.
`VBSaveImageToFile`	Loads image from specified file.
`VBDrawEyes`	Draws rectangle around eyes (this function is used only in Access sample).

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

6.4. Access

Access sample is located in `VLDemo.Access\` subdirectory of SDK. Directory contains following files:

File	Description
`VLDemo_msaccess.mdb`	VeriLook Sample
`ReadMe.txt`	Explains how to run sample
`CopyDLLs 98ME.bat`	Copies DLLs (`VLook.dll`, `VLVBP.dll`) to Windows system folder.

Code and database are located in the same file - `VLDemo_msaccess.mdb`. Access utilizes the same VeriLook parser as Visual Basic samples (`VLVBP.dll`). Sample contains modules and forms similar to Visual Basic 6 modules and forms.

Access sample is very similar to Visual Basic sample. Here are main differences:

- Sample work is controlled by controls which are on form (Visual Basic samples are controlled from menu)
- Rectangle around eyes is drawn using parser `DrawEyes` function.

Before running sample application ensure that following prerequisites are met:

- `VLook.dll` and `VLVBP.dll` are copied to Windows `System` folder.
- `VLCapturer.dll` is registered by running `bin\VLCapturerReg.bat`.

To run application – open MainForm form.

6.5. VeriLook Webcam Capturer (VLCapturer)

C#, Visual Basic and Access (VBA) are not designed to work with pointers so special ActiveX was created to work with web cameras.

`VLCapturer.dll` - webcam capturer ActiveX implementation. It must be registered in system before starting using it. Run `VLCapturerReg.bat` file from `\Bin` directory to perform registration.

`Capturer` (class implemented by `VLCapturer.dll`) has such interface (declarations of methods, events and properties are written using C# and Visual Basic 6.0 syntax):

Methods	
Method	Description

`EnumCaptureDevices`	
	```
	[C#]
	public void EnumCaptureDevices(
	        ref object devNames
	)

	[Visual Vasic]
	Sub EnumCaptureDevices(DevNames)
	```

`StartCapturing`	

	`[C#]`
	`public void StartCapturing(`
	`        string devName`
	`)`
	
	`[Visual Vasic]`
	`Sub StartCapturing(DevName As String)`

`StopCapturing`	`[C#]`
	`public void StopCapturing()`
	
	`[Visual Vasic]`
	`Sub StopCapturing()`

Properties	
Property	Description

`HorizontalFlipping`	`[C#]`
	`public bool HorizontalFlipping{`
	`        get;`
	`        set;`
	`}`
	
	`[Visual Vasic]`
	`Property HorizontalFlipping As Boolean`

Events	
Event	Description

`Image`	`[C#]`
	`public event`
	`AxVLCAPTURERLib._ICapturerEvents_ImageEventHandler`
	`Image`
	
	`[Visual Vasic]`
	`Event Image(Width As Long, Height As Long, Image)`