FingerCell 1.2 EDK

FingerCell 1.2 EDK

Table of Contents

1. Introduction 1
2. What's new 3
3. Fingerprint images 4
4. FingerCell library 5
    4.1. Library functions 5
    4.2. Error codes 7
    4.3. Initialization 9
        4.3.1. VFInitialize function 9
        4.3.2. VFFinalize function 9
    4.4. Contexts 10
        4.4.1. VFCreateContext function 12
        4.4.2. VFFreeContext function 12
    4.5. Parameters 13
        4.5.1. VFGetParameter function 18
        4.5.2. VFSetParameter function 19
        4.5.3. Additional functions 20
    4.6. Features extraction 22
        4.6.1. VFExtract function 22
    4.7. Features generalization 24
        4.7.1. VFGeneralize function 24
    4.8. Verification 26
        4.8.1. VFVerify function 26
    4.9. Identification 27
        4.9.1. VFIdentifyStart function 28
        4.9.2. VFIdentifyNext function 29
        4.9.3. VFIdentifyEnd function 30
    4.10. Matching threshold and similarity 30
    4.11. Matching details 31
    4.12. Fingerprint features 33
5. Sample applications 38
    5.1. Windows CE sample application 38
    5.2. Linux sample application 40

Chapter 1. Introduction

FingerCell EDK (Embedded Development Kit) is intended for embedded biometric system developers. The EDK incorporates the FingerCell fingerprint processing algorithm, which is especially designed to be used in embedded low-power and comparably low-CPU-power applications. The FingerCell algorithm provides high reliability of fingerprint identification in 1:1 and 1:N matching modes and has no constraints on interfaces or environments, thus enabling the use of most scanners and main processor types. The developer can easily and seamlessly integrate FingerCell into very diverse applications with various operating systems. FingerCell EDK is available either with FingerCell source code or with FingerCell library.

FingerCell EDK with library contains the following components:

General

- Sample fingerprint database;
- FingerCell EDK documentation.

Windows CE components:

- FingerCell library (for Microsoft Embedded Visual C++ 3.0);
- Sample application (project for Embedded Visual C++ 3.0, platform - Pocket PC).

Linux components:

- FingerCell library (for Arm-Linux GCC C compiler);
- ARM source code (ANSI C) of sample embedded application (project for Arm-Linux GCC C compiler);

FingerCell EDK with source code contains the following components:

- FingerCell source code:
  - Project for Arm-Linux GCC compiler
  - Project for MS Visual C++ 6.0
  - Project for MS Embedded Visual C++ 3.0
- FingerCell Algorithm and Source Code Description
- ARM source code of sample embedded application (project for Arm-Linux GCC C compiler)
- Sample application (project for Embedded Visual C++ 3.0, platform - Pocket PC)
- FingerCell EDK developers' guide.
- Sample fingerprint database

Later, where referenced:

`null` means `NULL`

`integer` - `INT`

Chapter 2. What's new

Version 1.2.0.0

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

Version 1.1.0.0

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

Chapter 3. Fingerprint images

Fingerprint image used by FingerCell library has to be an array of bytes of size width*height and pointer to the first element of this array has to be passed to libraries' functions. Lines of the image have to be stored in the array from top to bottom order. Next line must immediately follow the previous one (no padding). Each byte of the array corresponds to fingerprint image pixel (grayscale value). Value of 0 means black and value of 255 means white.

Chapter 4. FingerCell library

FingerCell library is a fingerprint recognition engine that you can use in your embedded devices.

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

When enrolling a user application can use features extraction functions that extracts features from fingerprint image (for more information see Fingerprint images and Features). Also 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.

Before using the library it has to be initialized. See Initialization and Contexts.

FingerCell library behavior is controlled through parameters.

4.1. Library functions

FingerCell library contains the following functions grouped by categories:

Registration	
VFRegistrationType	Returns registration type of FingerCell library
VFGenerateId	Generates registration id from serial number
VFRegister	Registers FingerCell library
Initialization	
VFInitialize	Initializes FingerCell library

VFFinalize	Uninitializes FingerCell library
Contexts	
VFCreateContext	Creates a context
VFFreeContext	Deletes the context
Parameters	
VFGetParameter	Retrieves parameter value
VFSetParameter	Sets parameter value
Features extraction	
VFExtract	Extracts features from fingerprint image
Features generalization	
VFGeneralize	Generalizes count features collections to single features collection
Verification	
VFVerify	Matches two features collections
Identification	
VFIdentifyStart	Starts identification with test features
VFIdentifyNext	Matches with sample features

VFIdentifyEnd	Ends identification

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

You can use `VFFailed` and `VFSucceeded` functions to determinate if the execution of the function failed or succeeded:

C:

```
#define VFFailed(result) ...
#define VFSucceeded(result) ...
```

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	FingerCell library is not initialized
`VFE_ARGUMENT_NULL`	-4	One of the required function arguments is null
`VFE_INVALID_ARGUMENT`	-5	One of the function arguments has an invalid value
`VFE_NOT_IMPLEMENTED`	-9	Function is not implemented

Parameters		
VFE_INVALID_PARAMETER	-10	Parameter identifier is invalid (unknown)
VFE_PARAMETER_READ_ONLY	-11	Parameter is read only
Features extraction		
VFE_ILLEGAL_IMAGE_RESOLUTION	-101	Specified image resolution is illegal
VFE_ILLEGAL_IMAGE_SIZE	-102	Specified image size is illegal
VFE_LOW_QUALITY_IMAGE	-103	Warning. Image quaity is low
FingerCell specific		
VFE_INVALID_MODE	-1000	Function called in invalid mode
Features		
VFE_INVALID_FEATURES_FORMAT	-3000	Features passed to the function has invalid format

You can use `VFErrorToString` and `VFResultToString` functions to get string that describes error and result. `VFCheckResult` function throws exception in case of the function result indicates failure. These functions are not part of FingerCell library. For C they are implemented in `FingerCell.h` and `FingerCellX.cpp` files.

C:

```
string VFErrorToString(INT error);
string VFResultToString(INT result);
void VFCheckResult(INT result);
```


4.3. Initialization

FingerCell library requires initialization to be performed before any function call and uninitialization to be performed after all function calls (except for contexts functions). This is performed using VFInitialize and VFFinalize functions.

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

Also you may not call initialization functions at all if you will not work with default context, only with yours custom context.

See Contexts for more information.

Example:

C:

```
// Main application function
{
        // Application initialization code
        VFInitialize();
        // Other application code
        VFFinalize();
        // Application uninitialization code
}
```

4.3.1. VFInitialize function

Creates default context by calling VFCreateContext function and initialized FingerCell library.

C:

```
INT VFINGER_API VFInitialize();
```

Return values: If succeeded return value means number of times function have been called before. If it first call to the function return value will be zero.

4.3.2. VFFinalize function

Destroys default context by calling VFFreeContext function and uninitializes FingerCell library

if call to the function corresponds to first call to VFInitialize function.

C:

```
INT VFINGER_API VFFinalize();
```

Return values: Return value means number of times function shold be more called (number of `VFInitialize` calls without `VFFinalize` calls).If FingerCell library was not initialized returns `VFE_NOT_INITIALIZED`.

4.4. Contexts

Context is a set of parameters and internal structures that FingerCell library functions use. They are created with VFCreateContext function and destroyed with VFFreeContext function.

Contexts enable different application parts to work with FingerCell library simultaneously. Inside one context no FingerCell functions should be called simultaneously because they are not guaranteed to be thread-safe. FingerCell functions called in different context are guaranteed to be thread-safe.

Parameters are set for the context. So you can use contexts not only to ensure that your application is thread safe, but to use different parameters in different situations also. For example you can perform features extraction for different scanners in different contexts with different set of parameters. For more information see Parameters.

Also particular FingerCell library functions should be called in particular order. If you have started identification (VFIdentifyStart) then you cannot call functions that work with internal matching structures (except for the VFIdentifyNext) such as VFSetParameter, VFGeneralize, VFVerify and VFIdentifyStart in the same context until you call VFIdentifyEnd. And you cannot call `VFIdentifyNext` and `VFIdentifyEnd` before you call `VFIdentifyStart` in the same context. In these situations functions will return `VFE_INVALID_MODE`.

Working from different threads:

C:

```
// First thread function
{
        // Create context
        HVFCONTEXT context = VFCreateContext();
        // Call FingerCell library functions, for example
        VFVerify(..., context);
        // Delete context
        VFFreeContext(context);
}
```

```
// Second thread function
{
        // Create context
        HVFCONTEXT context = VFCreateContext();
        // Call FingerCell library functions, for example
        VFIdentifyNext(..., context);
        // Delete context
        VFFreeContext(context);
}
```

Contexts with different parameters:

C:

```
HVFCONTEXT context1; // First context
HVFCONTEXT context2; // Second context
// Initialization function
{
        // Set parameters for default context
        VFSetParameter(..., NULL);
        VFSetParameter(..., NULL);
        // Create first context
        context1 = VFCreateContext();
        // Set parameters for first context
        VFSetParameter(..., context1);
        VFSetParameter(..., context1);
        // Create second context
        context2 = VFCreateContext();
        // Set parameters for second context
        VFSetParameter(..., context2);
        VFSetParameter(..., context2);
}

// Some application function
{
        HVFCONTEXT context;
        if (/* image from first scanner */)
                context = context1;
        else if(/* image from second scanner */)
                context = context2;
        else
                context = NULL; // default context
        // Call FingerCell library functions, for example
        VFExtract(..., context);
}
```



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

Example: Wrong functions call order:

C:

```
// Some application function
{
        //...
        VFIdentifyStart(...);
        for (...)
        VFIdentifyNext(...);
        VFVerify(...); // Error, returns VFE_INVALID_MODE
        VFIdentifyEnd(...);
        //...
        VFExtract(...);
        VFIdentifyNext(...); // Error, returns VFE_INVALID_MODE
}
```

4.4.1. VFCreateContext function

Creates context with default parameters.

C:

```
HVFCONTEXT VFINGER_API VFCreateContext();
```

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

4.4.2. VFFreeContext function

Deletes context created with VFCreateContext.

C:

```
INT VFINGER_API VFFreeContext(HVFCONTEXT context);
```

Parameters:

	context, Context	Context to delete

Return values: If context is null returns `VFE_ARGUMENT_NULL` else returns `VFE_OK`.

4.5. Parameters

Some FingerCell algorithm aspects are controlled through parameters. Parameters are retrieved and set for the specified context by VFGetParameter and VFSetParameter functions. Some parameters are read only (informational). If you will try to set a read only parameter `VFSetParameter` function will return `VFE_PARAMETER_READ_ONLY`. If you will pass an invalid parameter identifier to one of these functions it will return `VFE_INVALID_PARAMETER`.

Parameters can be of the following types:

Referenced as	Size (bytes)	`VF_TYPE_XXX` constant	C equivalent
Void		`VF_TYPE_VOID0`	
Byte	1	`VF_TYPE_BYTE1`	BYTE
Signed byte	1	`VF_TYPE_SBYTE2`	SBYTE
Word	2	`VF_TYPE_WORD3`	WORD
Short integer	2	`VF_TYPE_SHORT4`	SHORT
Double word	4	`VF_TYPE_DWORD5`	DWORD

Integer	4	VF_TYPE_INT6	INT
Boolean	4	VF_TYPE_BOOL10	BOOL
Char	1	VF_TYPE_CHAR20	CHAR
String	4	VF_TYPE_STRING100	CHAR*

To determine parameter type call VFGetParameter function with parameter identifier VFP_TYPE and value - needed parameter identifier. Also you may use VFGetParameterType function. Result of the function will be one of VF_TYPE_XXX constants.

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

When setting a parameter value pass the value casted to double word to VFSetParameter function.

There are functions VFGetXxxParameter and VFSetXxxParameter that work with particular parameter type.

For general parameters there are special functions VFGetXxx defined.

The following parameter identifiers are defined (grouped by categories):

Identifier	Value	Read only	Type	Description
General				
VFP_TYPE	0	x		See parameters types earlier
VFP_NAME	10	x	String	Name of the FingerCell library

VFP_VERSION_HIGH	11	x	Double word	Major version of FingerCell library
VFP_VERSION_LOW	12	x	Double word	Minor version of FingerCell library
VFP_COPYRIGHT	13	x	String	Copyright of FingerCell library
Features extraction				
VFP_EXTRACT_FEATURES	110		Integer	Obsolete, will be removed in the next version
VFP_RETURNED_IMAGE	10002		Integer	Specifies what image features extraction function will return. Can be one of the following:
VF_RETURNED_IMAGE_NONE	0	None image is returned - the image will be the same as passed to features extraction function		
VF_RETURNED_IMAGE_BINARIZED	100	Binarized image will be returned (default)		
VF_RETURNED_IMAGE_SKELETONIZED	200	Skeletonized image will be returned		
Features matching (Verification and Identification)				
VFP_MATCHING_THRESHOLD	200		Integer	Minimal similarity of two features collections that are identical. Must be not less that zero. See also Matching threshold
VFP_MAXIMAL_ROTATION	201		Integer	Maximal rotation of two features collection to each other. Must be in range VFDIR_0..VFDIR_180.

				See also information about directions in Features and Matching details
VFP_MATCH_FEATURES	210		Integer	Obsolete, will be removed in the next version
VFP_MATCHING_SPEED	220		Integer	Speed of features matching. Can be one of the following:
VF_MATCHING_SPEED_LOW	0	Low matching speed		
VF_MATCHING_SPEED_HIGH	256	High matching speed		
Features generalization				
VFP_GENERALIZATION_THRESHOLD.	300		Integer	Has the same meaning for features 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
FingerCell specific				
VFP_MODE	1000		Integer	Specifies mode in which FingerCell algorithm is operating (optimized parameter set) Can be one of the following:
VF_MODE_GENERAL	0	General		

VF_MODE_DIGITALPERSONA_UAREU	100	DigitalPersona U.are.U
VF_MODE_BIOMETRIKA_FX2000	200	BiometriKa FX2000
VF_MODE_KEYTRONIC_SECUREDESKTOP	300	Keytronic SecureDesktop
VF_MODE_IDENTIX_TOUCHVIEW	400	Identix TouchView
VF_MODE_PRECISEBIOMETRICS_100CS	500	PreciseBiometrics 100CS
VF_MODE_STMICROELECTRONICS_TOUCHCHIP	600	STMicroelectronics TouchChip
VF_MODE_IDENTICATORTECHNOLOGY_DF90	700	IdenticatorTechnology DF90
VF_MODE_AUTHENTEC_AFS2	800	Authentec AFS2
VF_MODE_AUTHENTEC_AES4000	810	Authentec AES4000
VF_MODE_ATMEL_FINGERCHIP	900	Atmel FingerChip
VF_MODE_BMF_BLP100	1000	BMF BLP100
VF_MODE_SECUGEN_HAMSTER	1100	SecuGen Hamster
VF_MODE_ETHENTICA	1200	Ethentica

VF_MODE_CROSSMATCH_VERIFIER300	1300	CrossMatch Verifier 300

4.5.1. VFGetParameter function

Retrieves specified parameter value for specified context.

C:

```
INT VFINGER_API VFGetParameter(INT parameter, VOID * value, HVFCONTEXT
context);
```

Parameters:

	parameter, Parameter	Parameter identifier to retrieve
[out]	value, Value	Pointer to variable that will receive parameter value
	Context, Context	Context to retrieve parameter from. Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns VFE_NOT_INITIALIZED.If parameter is invalid (unknown) returns VFE_INVALID_PARAMETER.If value is null returns VFE_ARGUMENT_NULL. For string parameters returns length of the string (not including the terminating null character).Otherwise returns VFE_OK.

Example:

C:

```
// Some application function
{
        CHAR *name;
        INT l;
        DWORD version;
        INT vfp_mode_type;
        INT mode;
```

```
        // Get FingerCell library name
        l = VFGetParameter(VFP_NAME, NULL, NULL);
        name = (CHAR*)malloc((l + 1) * sizeof(CHAR));
        VFGetParameter(VFP_NAME, name, NULL);
        printf(name);
        free(name);

        // Get FingerCell library major version
        VFGetParameter(VFP_VERSION_HIGH, version, NULL);
        printf("Version: %u.%u", version, HIWORD(version),
        LOWORD(version));

        // Determine parameter VFP_MODE type
        vfp_mode_type = VFGetParameter(VFP_TYPE, (VOID*)VFP_MODE, NULL);

        // returned value: VF_TYPE_INT
        // Get integer parameter VFP_MODE value
        VFGetParameter(VFP_MODE, &mode, NULL);
        printf("Mode: %d", mode);
}
```

4.5.2. VFSetParameter function

Sets specified parameter value for specified context.

C:

```
INT VFINGER_API VFSetParameter(INT parameter, DWORD value, HVFCONTEXT
context);
```

Parameters:

	parameter, Parameter	Parameter identifier to set
	value, Value	Parameter value to set
	context, Context	Context to set parameter to. Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is started returns `VFE_INVALID_MODE`.If parameter is invalid (unknown) returns `VFE_INVALID_PARAMETER`.Otherwise returns `VFE_OK`.

Example:

C:

```
// Some application function
{
        // Set VFP_EXTRACT_FEATURES parameter to VF_ALL_FEATURES
        VFSetParameter(VFP_EXTRACT_FEATURES, (DWORD) VF_ALL_FEATURES, NULL);
}
```

4.5.3. Additional functions

The following functions are not part of FingerCell library. They are implemented in `FingerCellX.h` and `FingerCellX.cpp` files.

`VFGetXxxParameter` functions retrieve parameters values of some type.

`VFSetXxxParameter` functions set parameters values of some type.

`VFGetXxx` functions retrieve general parameters values (`VFP_TYPE`, `VFP_NAME`, `VFP_VERSION_HIGH`, `VFP_VERSION_LOW`, `VFP_COPYRIGHT`).

C:

```
// Get
BYTE VFGetByteParameter(INT parameter, HVFCONTEXT context = NULL);
SBYTE VFGetSByteParameter(INT parameter, HVFCONTEXT context = NULL);
WORD VFGetWordParameter(INT parameter, HVFCONTEXT context = NULL);
SHORT VFGetShortParameter(INT parameter, HVFCONTEXT context = NULL);
DWORD VFGetDWordParameter(INT parameter, HVFCONTEXT context = NULL);
INT VFGetIntParameter(INT parameter, HVFCONTEXT context = NULL);
bool VFGetBoolParameter(INT parameter, HVFCONTEXT context = NULL);
TCHAR VFGetCharParameter(INT parameter, HVFCONTEXT context = NULL);
string VFGetStringParameter(INT parameter, HVFCONTEXT context = NULL);
// Set
void VFSetByteParameter(INT parameter, BYTE value, HVFCONTEXT context = NULL);
void VFSetSByteParameter(INT parameter, SBYTE value, HVFCONTEXT context =
NULL);
void VFSetWordParameter(INT parameter, WORD value, HVFCONTEXT context = NULL);
void VFSetShortParameter(INT parameter, SHORT value, HVFCONTEXT context =
```

```
NULL);
void VFSetDWordParameter(INT parameter, DWORD value, HVFCONTEXT context =
NULL);
void VFSetIntParameter(INT parameter, INT value, HVFCONTEXT context = NULL);
void VFSetBoolParameter(INT parameter, bool value, HVFCONTEXT context = NULL);
void VFSetCharParameter(INT parameter, TCHAR value, HVFCONTEXT context =
NULL);
void VFSetStringParameter(INT parameter, const string& value, HVFCONTEXT
context = NULL);
// Get general
INT VFGetParameterType(INT parameter);
string VFGetName();
DWORD VFGetVersionHigh();
DWORD VFGetVersionLow();
string VFGetCopyright();
```

Parameters:

	parameter, Parameter	Parameter identifier to retrieve or set
	value, Value	Parameter value to set
	context, Context	context retrieve or set parameter value for (by default - null, default context)

Return values: `VFGetXxxParameter` functions return specified parameter value.`VFGetXxx` functions return corresponding general parameter value.Other functions return nothing.

Exceptions: All functions raise exception in case of error.

Example:

C:

```
// Some application function
{
        string name;
        DWORD version;
        INT vfp_mode_type;
        INT mode;
```

```
        // Get FingerCell library name
        name = VFGetStringParameter(VFP_NAME);
        // or
        name = VFGetName();
        printf(name);
        // Get FingerCell library major version
        version = VFGetDWordParameter(VFP_VERSION_HIGH);
        // or
        version = VFGetVersionHigh();
        printf("Version: %u.%u", version, HIWORD(version),
        LOWORD(version));

        // Determine parameter VFP_MODE type
        vfp_mode_type = VFGetParameterType(VFP_MODE);
        // returned value: VF_TYPE_INT

        // Get integer parameter VFP_MODE value
        mode = VFGetIntParameter(VFP_MODE);
        printf("Mode: %d", mode);
}
```

4.6. 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.

4.6.1. VFExtract function

Extracts features from fingerprint image in the specified context.

Image has to be an array of bytes of size width*height and pointer to the first element of this array has to be passed to this function. Image resolution has to be passed to the function. Function resizes image to resolution of `VF_IMAGE_RESOLUTION` dpi (dots per inch) internally. Filtered image that is returned by the function is resized back to original resolution. Features returned by the function have resolution of `VF_IMAGE_RESOLUTION` dpi, so if you wish to display features on the image you have to draw features on the image resized to `VF_IMAGE_RESOLUTION` dpi.

Features have to be an array of bytes and pointer to the first element of the array has to be passed to this function (in Visual Basic case - image array are passed to functions). Number of bytes occupied by features in the array will be returned by the function in size (Size) parameter. If array

size is less than needed then behavior of the function is undefined. To ensure that array is large enough set its size to at least VF_MAX_FEATURES_SIZE. For more information see Features.

The function uses features extraction and FingerCell specific parameters.

C:

```
#define VF_IMAGE_RESOLUTION 250
#define VF_FEATURES_RESOLUTION 500
#define VF_MAX_FEATURES_SIZE 10000
INT VFINGER_API VFExtract(INT width, INT height, BYTE * image, INT resolution,
BYTE * features, DWORD * size, HVFCONTEXT context);
```

Parameters:

	width, Width	Fingerprint image width. After resize have to be in range from VF_MIN_IMAGE_DIMENSION to VF_MAX_IMAGE_DIMENSION
	height, Height	Fingerprint image height. After resize have to be in range from VF_MIN_IMAGE_DIMENSION to VF_MAX_IMAGE_DIMENSION
[in/out]	image, Image	Fingerprint image to extract features from. After execution of the function contains filtered image
	resolution, Resolution	Resolution of the fingerprint image (in dots per inch). Must be in range from VF_MIN_IMAGE_RESOLUTION to VF_MAX_IMAGE_RESOLUTION
[out]	features, Features	After execution of the function contains features extracted from fingerprint image
[out]	size, Size	After execution of the function contains size of the features in bytes.
	context, Context	Context to perform features extraction in. Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If fingerprint image width or height is not in legal range returns `VFE_ILLEGAL_IMAGE_SIZE`.If resolution is not in legal range returns `VFE_ILLEGAL_IMAGE_RESOLUTION`.If image, features or size is null returns `VFE_ARGUMENT_NULL`.Otherwise performs features extraction and returns either `VFE_OK` or `VFE_LOW_QUALITY_IMAGE` (if fingerprint image quality is low). `VFE_LOW_QUALITY_IMAGE` is only a warning. Calling application can either ignore it or ask the user to rescan the fingerprint.

Example:

C:

```
// Extraction function
{
        INT width, height;
        BYTE *image;
        INT resolution;
        BYTE features[VF_MAX_FEATURES_SIZE];
        DWORD size;

        // Load the image from file or get the image from scanner
        // ...
        VFExtract(width, height, image, resolution, features, &size, NULL);
}
```

4.7. Features generalization

You can use features generalization to increase quality of the recognition. Generalization performs conjunction of several features collections to one collection, validates each feature and removes noisy features. You can use features generalization during enrollment. To obtain 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.

4.7.1. VFGeneralize function

Performs generalization of features collections in the specified context. Currently generalization can be performed only for `VF_GENERALIZE_COUNT` features collections.

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

C:

```
#define VF_GENERALIZE_COUNT 3
INT VFINGER_API VFGeneralize(INT count, const BYTE * const *gen_features, BYTE
* features, DWORD * size, HVFCONTEXT context);
```

Parameters:

	count, Count	Count of features collections to generalize
[in]	gen_features, GenFeatures	Array of features collections to generalize
[out]	features, Features	After execution of the function contains generalized features
[out]	size, Size	After execution of the function contains size of generalized features in bytes.
	context, Context	Context to perform features generalization in. Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is started returns `VFE_INVALID_MODE`.If count of features collections is other that `VF_GENERALIZE_COUNT` returns `VFE_INVALID_ARGUMENT`.If features collections are null returns `VFE_ARGUMENT_NULL`.If one of the passed features collections has invalid format returns `VFE_INVALID_FEATURES_FORMAT`.If features collections cannot be generalized returns `VFE_FAILED`.Otherwise return index of features collection on which base features generalization has been performed.

Example:

C:

```
// Generalization function
{
```



```
        BYTE *feats[3];
        BYTE features[VF_MAX_FEATURES_SIZE];
        DWORD size;
        feats[0] = /*obtain first fingerprint features*/;
        feats[1] = /*obtain second fingerprint features*/;
        feats[2] = /*obtain third fingerprint features*/;
        if (VFSucceeded(VFGeneralize(3, feats, features, &size, NULL))
                printf("Generalization succeeded");
        else
                printf("Generalization failed");
}
```

4.8. Verification

You can use verification to determinate if two features collections are of the same finger. It uses `VFP_MATCHING_THRESHOLD` parameter (See Matching threshold). To obtain features from fingerprint image use features extraction functions. Also you may use features generalization functions to increase recognition reliability.

Use VFVerify function to perform verification.

4.8.1. VFVerify function

Performs two features collections verification in the specified context.

Pass pointer to matching details structure that will receive details of features collections matching (set size (Size) member before calling the function to actual size of the structure). Pass null if you are not interested in matching details. For more information see Matching details.

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

C:

```
INT VFINGER_API VFVerify(const BYTE * features1, const BYTE * features2,
VFMATCHDETAILS * md, HVFCONTEXT context);
```

Parameters:

[in]	features1, Features1	First fingerprint features

[in]	features2, Features2	Second fingerprint features
[in/out]	md, MD	After execution of the function contains details of features collections matching
	context, Context	Context to perform verification in. Null for default context

Return values: If context is null and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is started returns `VFE_INVALID_MODE`.If one of the features collections is `null` returns `VFE_ARGUMENT_NULL`.If one of the passed features collections has invalid format returns `VFE_INVALID_FEATURES_FORMAT`.If insufficient memory then returns `VFE_OUT_OF_MEMORY`.If features collections similarity is high enough (see `VFP_MATCHING_THRESHOLD` in Parameters) returns `VFE_OK` (the same finger features collections). Otherwise returns `VFE_FAILED`.

Example:

C:

```
// Verification function
{
        BYTE *features1, *features2;
        VFMATCHDETAILS md;
        BOOL result;
        features1 = /*obtain first fingerprint features*/;
        features2 = /*obtain second fingerprint features*/;
        md.size = sizeof(md);
        result = VFSucceeded(VFVerify(features1, features2, &md, NULL));
        if (result)
                printf("Same finger. Similarity: %d", md.similarity);
        else
                printf("Different fingers. Similarity: %d", md.similarity);
}
```

4.9. Identification

Use identification to identify fingerprint in the database.

First start identification with unknown fingerprint (test) features. Use VFIdentifyStart function.

Then walk through all database features (sample features) and match them with test features (with VFIdentifyNext function) until matched (function returns VFE_OK) or end of the database passed. It uses VFP_MATCHING_THRESHOLD parameter (see Matching threshold).

You may also use G to increase speed of the identification: match first sample features which G is equal to test features G; then sample features which G difference with test features is 1, then with G difference 2 and so on. It is a quite high probability that fingerprint will be identified during first matches if it is in the database. See also Features.

End the identification (VFIdentifyEnd function).

To obtain features for identification use features extraction function (for enrollment in the database you may also use features generalization functions).

Example:

C:

```
// Identification function
{
        BYTE *test_features;
        BYTE *sample_features;
        BOOL found;

        test_features = /*obtain features of fingerprint to identify*/;
        VFIdentifyStart(test_features, NULL);
        found = FALSE;
        for (/* walk through database */)
        {
                sample_features = /*some features from the database*/;
                if (VFSucceeded(VFIdentifyNext(sample_features, NULL, NULL)))
                {
                        found = TRUE;
                        break;
                }
        }
        VFIdentifyEnd(NULL);
        if (found)
                printf("Fingerprint found in the database");
        else
                printf("Fingerprint not found in the database");
}
```

4.9.1. VFIdentifyStart function

Starts identification with specified test features in specified context.

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

C:

```
INT VFINGER_API VFIdentifyStart(const BYTE * test_features, HVFCONTEXT
context);
```

Parameters:

[in]	test_features, TestFeatures	Test features
	context, Context	Context to start identification in Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is started returns `VFE_INVALID_MODE`.If test features collection has invalid format returns `VFE_INVALID_FEATURES_FORMAT`.If test features are null returns `VFE_ARGUMENT_NULL`.If insufficient memory then returns `VFE_OUT_OF_MEMORY`.

4.9.2. VFIdentifyNext function

Matches sample features with test features in specified context.

Pass pointer to matching details structure that will receive details of features collections matching (set size (Size) member before calling the function to actual size of the structure). Pass null if you are not interested in matching details. For more information see Matching details.

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

C:

```
INT VFINGER_API VFIdentifyNext(const BYTE * sample_features, VFMATCHDETAILS *
md, HVFCONTEXT context);
```

Parameters:

[in]	sample_features, SampleFeatures	Sample features
[in/out]	md, MD	After execution of the function contains details of features collections matching.
	Context, Context	Context to perform features matching in. Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is not started returns `VFE_INVALID_MODE`.If sample features are null returns `VFE_ARGUMENT_NULL`.If test features collection has invalid format returns `VFE_INVALID_FEATURES_FORMAT`.If features collections similarity is high enough (see `VFP_MATCHING_THRESHOLD` in Parameters) returns `VFE_OK` (the same finger features collections). Otherwise returns `VFE_FAILED`.

4.9.3. VFIdentifyEnd function

Ends the identification started with VFIdentifyStart function in specified context.

C:

```
INT VFINGER_API VFIdentifyEnd(HVFCONTEXT context);
```

Parameters:

	context, Context	Context to end identification in. Null for default context

Return values: If context is `null` and FingerCell library is not initialized returns `VFE_NOT_INITIALIZED`.If identification is not started returns `VFE_INVALID_MODE`.

4.10. Matching threshold and similarity

FingerCell features matching algorithm provides value of features collections similarity as a result. It can be obtained in matching details. The higher is similarity, the higher is probability that features collections are obtained from the same finger fingerprints.

You can set matching threshold - the minimum similarity value that verification and identification functions accept for the same finger fingerprints. You can set the matching threshold using `VFP_MATCHING_THRESHOLD` parameter (`VFP_GENERALIZATION_THRESHOLD` for features generalization).

Matching threshold is linked to false acceptance rate (FAR, different fingers fingerprints erroneously accepted as the of the same finger) of matching algorithm. The higher is threshold, the lower is FAR and higher FRR (false rejection rate, same finger fingerprints erroneously accepted as different fingers fingerprints) and vice a versa. You can use `VFMatchingThresholdToFAR` and `VFFARToMatchingThreshold` functions to convert, matching threshold to FAR in percents and vice a versa. Only values of and for FAR between 1% and 0.001% can be calculated more or less correctly. All other values are calculated very approximately. These functions are not part of FingerCell library; they are implemented in `FingerCellX.h` and `FingerCellX.cpp` files.

C:

```
double VFMatchingThresholdToFAR(INT th);
INT VFFARToMatchingThreshold(double f);
```

4.11. Matching details

Matching details describes relationship between two features collections determined during verification or identification. Matching details are defined as structure, pointer to which you can pass to verification or identification functions. It can be one of the following structures. When passing to the function set size (Size) member to size of actual structure and cast pointer to the structure to pointer to the first structure. See also Features.

C:

```
#define VF_MAX_MINUTIA_COUNT 1024
typedef struct _VFMATCHDETAILS
{
        DWORD size;
        INT similarity;
        INT rotation;
        INT trans_x;
        INT trans_y;
} VFMATCHDETAILS;

typedef struct _VFMATCHDETAILSEX
{
        DWORD size;
        INT similarity;
```

```
        INT rotation;
        INT trans_x;
        INT trans_y;
        INT mm_count;
        SHORT mm[2][VF_MAX_MINUTIA_COUNT];
} VFMATCHDETAILSEX;
```

Members:

size, Size	Size of the structure. Set this member before calling a function
similarity, Similarity	Two features collections similarity value. The bigger is value the higher is similarity. See also Matching threshold and similarity
rotation, Rotation	Rotation of two features collections to each other. It is an angle in range [`0`, `VFDIR_360`). See also information about directions in Features
Trans_x, TransX	Translation between two features collections along X axis
Trans_y, TransY	Translation between two features collections along Y axis
mm_count, MMCount	Count of minutiae common for two features collections in matched minutiae array
mm, MM	Matched minutiae array (common minutiae for two features collections). First index of the array means first (0) or second (1) features collection, second - index of minutia in features collection. See also Features

Example:

C:

```
//Identification function
{
        BYTE *test_features;
        BYTE *sample_features;
```



```
        VFMATCHDETAILS md;

        //...
        VFIdentifyStart(test_features, NULL);
        md.size = sizeof(md);
        for (...)
        {
                VFIdentifyNext(sample_features, &md, NULL);
                printf("Similarity: %d", md.similarity);
        }
        VFIdentifyEnd(NULL);
}

// Verification function
{
        BYTE *features1, *features2;
        VFMATCHDETAILSEX md;
        INT I;

        // ...
        md.size = sizeof(md);
        VFVerify(features1, features2, (VFMATCHDETAILS)&md, NULL);
        for (i = 0; i < md.mm_count; i++)
                printf("Matched minutia %d: index in first features - %d"
                "; index in second features - %d\n", i, md.mm[0][i],
md.mm[1][i]);
}
```

4.12. Fingerprint features

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

Features are stored in array of bytes. Size of the array will never exceed `VF_MAX_FEATURES_SIZE`.

You can use `VFDecompressFeatures` and `VFCompressFeatures` functions to decompress features to and compress features from structure. These functions are not part of FingerCell library. They are implemented in `FingerCellX.h` and `FingerCellX.cpp` files. Also you may use `CVFFeatures` class (`VFFeatures.h` and `VFFeatures.cpp` files in sample application) that encapsulates features, compression and decompression. You can use the class or compression and decompression functions to implement manual features editing in your application.

C:

```
typedef enum _VFSingularPointType
{
        vfsptUnknown = 0,
        vfsptCore = 1,
        vfsptDoubleCore = 2,
        vfsptDelta = 3
} VFSingularPointType;

typedef struct _VFSingularPoint
{
        INT X;
        INT Y;
        VFSingularPointType T;
        BYTE D;
} VFSingularPoint;

typedef enum _VFMinutiaType
{
        vfmtUnknown = 0,
        vfmtEnd = 1,
        vfmtBifurcation = 2
} VFMinutiaType;

typedef struct _VFMinutia
{
        INT X;
        INT Y;
        VFMinutiaType T;
        BYTE D;
        BYTE C;
        BYTE G;
} VFMinutia;

// Features compression
INT VFINGER_API VFFeatSet(BYTE g, INT mCount, const VFMinutia * m, INT
spCount, const VFSingularPoint * sp, INT boWidth, INT boHeight, const BYTE *
bo, BYTE * features);
// Features decompression
INT VFINGER_API VFFeatGetG(const BYTE * features);
INT VFINGER_API VFFeatGetMinutiaCount(const BYTE * features);
INT VFINGER_API VFFeatGetMinutiae(const BYTE * features, VFMinutia * m);
INT VFINGER_API VFFeatGetSPCount(const BYTE * features);
INT VFINGER_API VFFeatGetSP(const BYTE * features, VFSingularPoint * sp);
INT VFINGER_API VFFeatGetBOSize(const BYTE * features,
```



```
INT * pWidth, INT * pHeight);
INT VFINGER_API VFFeatGetBO(const BYTE * features, BYTE * bo);
```

All functions take features (Features) argument as compressed features and/or m (M) argument as minutia array (mCount (MCount) is length of the array), sp (SP) argument as singular point array (spCount (SPCount) is length of the array), bo (BO) argument as blocked orientation image (in similar format as fingerprint image; boWidth (BOWidth) and boHeight (BOHeight) are accordingly width and height of blocked orientations image). `VFFeatSet` function returns size of compressed features.

Features consist of:

- G - obtained using `VFFeatGetG` function
- Minutiae - obtained using `VFFeatGetMinutiae` function (number of minutiae obtained using `VFGetMinutiaCount` function)
- Singular points - obsolete, obtained using `VFFeatGetSP` function (number of singular points obtained using `VFGetSPCount` function). FingerCell does not extract singular points
- Blocked orientations - obsolete, obtained using `VFFeatGetBO` function (size of blocked orientations obtained using `VFGetBOSize` function). FingerCell does not extract blocked orientations

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

Minutiae are points in fingerprint image where finger ridges end or separate. Each minutia is a structure with the following members: X, Y - coordinates in pixels, T - type, D - direction, C - curvature, G - g.

Singular points are points in fingerprints image where finger ridges screw. Each singular point is a structure with the following members: X, Y - coordinates in pixels, T - type, D - direction.

Each minutia and singular point has x and y coordinates (from top-left corner of the image) and direction.

Direction is value in range [`VFDIR_0`, `VFDIR_360`) - byte value. To convert it to degrees multiply by 180 and divide by `VFDIR_180` and vice a versa to convert degrees to direction. Also you may use `VFDirToDeg` and `VFDegToDir` functions. To covert them to radians and vice a versa use `VFDirToRad` and `VFRadToDir` functions. The following constants are defined:

VFDIR_0	0	0°
VFDIR_45	30	45°
VFDIR_90	VFDIR_45 * 2	90°
VFDIR_135	VFDIR_45 * 3	135°
VFDIR_180	VFDIR_45 * 4	180°
VFDIR_225	VFDIR_45 * 5	225°
VFDIR_270	VFDIR_45 * 6	270°
VFDIR_315	VFDIR_45 * 7	315°
VFDIR_360	VFDIR_45 * 8	360°
VFDIR_UNKNOWN	127	Unknown direction
VFDIR_BACKGROUND	255	Background

Blocked orientations are fingerprint image divided in blocks of VF_BLOCK_SIZE * VF_BLOCK_SIZE pixels. Value of each block is ridges orientation of fingerprint image in that block. Can be in range [VFDIR_0, VFDIR_180) or VFDIR_UNKNOWN or VFDIR_BACKGROUND - byte value.

C:

```
// Conversion from VFDIR_XXX to degrees and vice a versa
#define VFDirToDeg(dir) ...
#define VFDirToDegF(dir) ...
#define VFDegToDir(deg) ...

// Conversion from VFDIR_XXX to radians and vice a versa
#define VFDirToRad(dir) ...
```



```
#define VFRadToDir(a) ...

// Orientation stuff
#define VFIsBadArea(orient) ...
#define VFIsGoodArea(orient) ...
#define VFTheOrient(orient) ...
#define VFIsUnknown(orient) ...
#define VFIsOrient(orient) ...
```


Chapter 5. Sample applications

5.1. Windows CE sample application

Sample application was created for Embedded Visual C++ 3.0 (platform - Pocket PC).

Project contains such files:

File	Description
`FCDemo.vcp`	Embedded Visual C++ 3.0 project
`FCDemo.vcw`	Embedded Visual C++ 3.0 project workspace
`FingerCellX.h` `FingerCellX.cpp`	FingerCell additional functions (for more information please see Additional functions)
`VFFeatures.cpp` `VFFeatures.h`	Defines and implements class for fingerprint features manipulations
`FPDatabase.cpp` `FPDatabase.h`	Defines and implements class for fingerprint records manipulation

`Image.cpp`
`Image.h`
`2dbytearray.cpp`
`2dbytearray.h`
	Modules for image manipulations and loading from BMP and TIF files.

`FingerCellView.cpp` `FingerCellView.h`	Defines and implements application document view class

FingerCellDoc.cpp FingerCellDoc.h	Defines and implements application document class
MainFrm.cpp MainFrm.h	Defines and implements application main form class
LogerInterface.cpp LogerInterface.h	Defines interface for logging
LogView.cpp LogView.h	Defines and implements log view class
FingerCell.cpp FingerCell.h	Defines and implements application class
SettingsDialog.cpp SettingsDialog.h	Defines and implements settings dialog class
StdAfx.cpp StdAfx.h	Module for precompiled headers
Utils.cpp Utils.h	Module for time calculations
FingerCell.rc newres.h Resource.h	Application resources information
\res	Folder which contains sample resources

Important
Before compiling application with Embedded Visual C++ please ensure that stack size will be 386kb or more (please review project options).

Sample applications has four possible working modes (user must choose mode when starts application):

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

According mode demonstration program will call different FingerCell library functions.

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

5.2. Linux sample application

The Linux demonstrational program is intended to demonstrate the basic use of the FingerCell library. It is very simple to make it work in as many different environments as possible. The main part of the code is in the file `demo.c`. A TIFF (Tagged Image File Format) reader is implemented in the file `Image.cpp`. The program reads image files from the ./samples directory and performs processing on the images with the help of FingerCell functions. The results of the processing (the time it takes to process each image, if the images match or not etc.) are printed to the console.

File	Description
`FCDemo/`	FingerCell demo application
`Makefile`	The program can be compiled by typing `make clean; make`
`array.h`	A header file for `array.c`
`wrap.h`	A header file for `wrap.c`
`image.h`	A header file for `image.c`
`fc_console_demo`	The executable demo
`wrap.c`	A simple wraper to `image.c`
`image.c`	A TIFF file reader
`array.c`	A generic allocation of 2-dimensional arrays (used by `image.c`)
`demo.c`	The main code for the demo
`readme`	ReadMe file
`include/`	
`FingerCell.h`	A header, which is needed to use the FingerCell functions
`lib/`	
`libFingerCell.a`	The FingerCell library (gets compiled into `fc_console_demo`)

`samples/`	The directory holding a few files to process

