Manual SDK v. 3.6

WAC Fingerprint SDK

WAC RESEARCH CO.,LTD.
11/587 Moo.10, Ladpraowanghin, Ladprao,
Bangkok 10230
Tel: 025303809-10, 025381038, 025399352
Fax:025383098
E-Mail: sales@wacinfotech.com
www.wacinfotech.com

Introduction

The Software Development Kit (SDK) is intended for creation of biometric applications based on the fingerprint recognition. It gives the developers the ability to enroll, verify and identify the fingerprint templates.

This documentation assumes that a developer has a common impression about biometry and its applications. Some important intentions are explained in the Concept terms.

To get a brief notion of biometric application design based on the application programming interface (API) you should look at the Application types.

The detailed explanation of API functions and their parameters: SDK functions in alphabetical order.

Concept terms

Some concept terms used in this manual and everywhere in biometry are denoted in this section.

- The process of initial template construction is called *enrollment*. Some fingerprint images are collected through the sensor device, their main properties are extracted and the result is stored somewhere by an application for further comparison (matching).

- The comparison of templates can be organized as one-to-one or one-to-many matching. The first case is called Verification and the second one is called Identification. *Verification* is used whenever an application needs to check if a particular template looks like the previously built template.

The *Identification* allows to find a group of templates among the source set, that are mostly "similar" to the specified template. The result of this search can be either empty or can contain some templates.

- Any kind of biometric authentication can be expressed only in terms of probability. The significant reason of such situation is the fact that you cannot obtain two absolutely identical fingerprints (or any other biometric measurements of a human) gathered in different sensor touches.

The main score used herein shows the "trusted level" of any authentication operation and is called False Accepting Rate (*FAR*). It denotes the probability that the source template falsely match the presented template. If a particular *FAR* value is equal P, it means that an actual False Accepting Rate is calculated as P/(2 exp 31 -1). The larger value implies the "softer" result. Instead of stochastic form a user can specify the FAR value in the terms of Numerical form, which is designated as FTR_FARN. This value denotes the calculated internal measure. The lower value reflects the “softer’ result.

Application types

A typical application execution flow depends on type of an application. The SDK introduces two types of application:

- Application requires interaction with a user. The main task of that application is to capture images from the attached sensor and to create templates appropriately to the specified

purpose. Such application can optionally perform any authentication operations. The API caller gets a responsibility for writing interaction with a user in the form of prompts when it is necessary to touch a sensor and when to take a finger from the sensor surface. Definitely, an appropriate call must be issued to declare the usage of a sensor. Interactive application example shows a typical function calls sequence.

- Application does not require any interaction with a user. An application that uses only the matching mechanism and doesn't require user interaction has a simpler structure. Such application employs only the series of FTRSetBaseTemplate and FTRIdentify function calls and implements the core of authenticating center. A skeleton of Non-interactive application example depicts the architecture of a typical authentication algorithm.

Interactive application example

```
// Application requires interaction with a user. if( FTRInitialize() != FTR_RETCODE_OK )

return;

// Use the specified sensor.
if( FTRSetParam( FTR_PARAM_CB_FRAME_SOURCE, FSD__USB ) == FTR_RETCODE_OK ){


// Get the frame dimensions.
FTRGetParam( FTR_PARAM_IMAGE_WIDTH, &dwWidth );
FTRGetParam( FTR_PARAM_IMAGE_HEIGHT, &dwHeight );
FTRGetParam( FTR_PARAM_IMAGE_SIZE, &dwSize );


pImage = malloc( dwSize ); // Reserve memory space for an image.

// Optional. Set the maximum number of frames in a template. This call must precede
// the FTRGetParam(FTR_PARAM_MAX_TEMPLATE_SIZE, … ) call.
FTRSetParam( FTR_PARAM_MAX_MODELS, 3 );


// Get the maximum template size in bytes.
FTRGetParam( FTR_PARAM_MAX_TEMPLATE_SIZE, &dwTempSize );
FTR_DATA Template;
Template.pData = malloc( dwTempSize ); // Reserve memory for a template.


// Establish the user interaction callback function.
// Note, that the cbUserSuppliedFunc must be declared accordingly to the FTR_CB_STATE_CONTROL,
// see details in the FTRAPI.h header file.
FTRSetParam( FTR_PARAM_CB_CONTROL, cbUserSuppliedFunc );


// Build a template for the verification purpose.
FTR_USER_CTX myContext;
FTREnroll( myContext, FTR_PURPOSE_ENROLL, &Template );
```

// Verify if a user matches the specified template with the FAR = 0.05 BOOL bIsVerified; FTRSetParam(FTR_PARAM_MAX_FAR_REQUESTED, 107374182); // 107374182 / (2**31 -1) = 0.05 if(FTRVerify(myContext, &Template, &bIsVerified, NULL) == FTR_RETCODE_OK){

if(bIsVerified){

// Proceed a match!
}
else{

// Match was not detected. } } }

FTRTerminate();

Non-interactive application example

// Application does not need to interact with a user. if(FTRInitialize() != FTR_RETCODE_OK) return;

DWORD dwBufCount = 4; // Number of buffered templates. FTR_DATA Templates[dwBufCount]; FTR_IDENTIFY_RECORD IdRecords[dwBufCount]; FTR_IDENTIFY_ARRAY SourceData;

// Optional. Set the maximum number of frames in a template. This call must precede // the FTRGetParam(FTR_PARAM_MAX_TEMPLATE_SIZE, …) call. FTRSetParam(FTR_PARAM_MAX_MODELS, 3);

// Get the maximum template size in bytes.
DWORD dwTempSize;
FTRGetParam(FTR_PARAM_MAX_TEMPLATE_SIZE, &dwTempSize);

// Dynamically allocate memory for the source templates.
LPSTR pBuffer;
pBuffer = malloc(dwBufCount * dwTempSize);

// Initialize source data.
SourceData.pMembers = IdRecords;
SourceData.TotalNumber = dwBufCount;
for(i = 0; i < dwBufCount; i++){

IdRecords[i].pData = Templates + i; Templates[i].pData = pBuffer + dwTempSize * i; }

// Initialize memory for the matching data.
FTR_MATCHED_RECORD mr[3];
FTR_MATCHED_ARRAY ma;
ma.TotalNumber = sizeof(mr) / sizeof(FTR_MATCHED_RECORD);
ma.pMembers = mr;

// Identify with the FAR = 0.01; (21474837 / 2147483647 = 0.01).
FTRSetParam(FTR_PARAM_MAX_FAR_REQUESTED, 21474837);

// Prepare the template to look for across the source template array.

if(FTRSetBaseTemplate(&Template) == FTR_RETCODE_OK){ FTRAPI_RESULT RetCode; DWORD dwMatchCnt = 0; // This is important! The caller must properly initialize the // total number of matched records before calling the FTRIdentify function within the cycle.

while(LoadNextSourcePortion(&SourceData) > 0){ // Search through the currently available portion of templates. if((RetCode = FTRIdentify(&SourceData, &dwMatchCnt, &ma)) != FTR_RETCODE_OK) break;

}

if(RetCode == FTR_RETCODE_OK && dwMatchCnt > 0){
// Do something with matched records.
for(i = 0; i < dwMatchCnt; i++)

UseMatch(ma.pMembers[i].KeyValue); } }

FTRTerminate();

State callback function

Every biometric application that employs a capturing frame service needs to organize interaction with a user. This interaction is implemented in the form of following advises:

• touch -means that it is the right moment to touch scanner surface;

• take off -a user must take his/her finger from the device surface.

Captured frames represent another kind of information that can be shown to a user.

The described information exchange is done over an application-supplied callback function with the prototype

```
void (FTR CBAPI *FTR CB STATE CONTROL)(
FTR USER CTX Context, // [in] user-defined context information
FTR STATE StateMask, // [in] an arguments bit mask
FTR RESPONSE *pResponse, // [in, out] progress data or response from user
FTR SIGNAL Signal, // [in] signal value
FTR BITMAP PTR pBitmap // [in] a pointer to the bitmap to be displayed
```

```
);
```

Parameters

Context

[in] user-defined context information. This is the value passed via the *UserContext* parameter in FTREnroll and FTRVerify functions.

StateMask

[in] a bit mask indicating what arguments are provided. This mask can be a combination of the following constants:

State	Meaning
FTR_STATE_FRAME_PROVIDED	The *pBitmap* parameter provided.
FTR_STATE_SIGNAL_PROVIDED	The *Signal* parameter provided.

pResponse

[in] pointer to the FTR_PROGRESS structure containing information on various aspects of frame capturing progress.
To gain access to the structure members a caller must cast the value passed in this parameter to a FTR_PROGRESS
pointer.
[out] pointer to a value indicating the required action from the calling API function. Any constant from the following list
can be used. It is the application responsibility to set the appropriate value of this parameter.

Response code	Meaning
FTR_CANCEL	The calling function must return control as quickly as possible. The caller returns the FTR_RETCODE_CANCELED_BY_USER value.
FTR_CONTINUE	The calling function can continue execution.

Signal

[in] this value should be used to interact with a user. Any separate constant from the following table is passed to a callback function.

Signal value	Description

FTR_SIGNAL_TOUCH_SENSOR	Invitation for touching the fingerprint scanner surface.
FTR_SIGNAL_TAKE_OFF	Proposal to take off a finger from the scanner surface.
FTR_SIGNAL_FAKE_SOURCE	Notification on the Fake Finger Detection (FFD) event.

pBitmap
[in] a pointer to the bitmap to be displayed. The usage of this parameter is optional.

SDK functions listed in alphabetical order

The SDK functions listed in alphabetical order.

Function	Description
FTRCaptureFrame	Gets an image from the current frame source.
FTREnroll	Creates the fingerprint template for the desired purpose.
FTRGetParam	Gets the value of the specified parameter.
FTRIdentify	Compares the base template against a set of source templates.
FTRInitialize	Activates the SDK interface.
FTRSetBaseTemplate	Installs a template as a base for identification process.
FTRSetParam	Sets the indicated parameter value.
FTRTerminate	Deactivates the API.
FTRVerify	Verifies a captured image against the specified template.

FTRCaptureFrame

The FTRCaptureFrame function gets an image from the current frame source.

```
FTRAPI_RESULT FTRCaptureFrame(
  FTR_USER_CTX UserContext,  // [in] optional value
  void *pFrameBuf            // [in] pointer to the frame buffer
);
```

Parameters

UserContext

[in] -optional caller-supplied value that is passed to callback functions. This value is provided for convenience in application design.

pFrameBuf

[in] points to a buffer large enough to hold the frame data. The size of a frame can be determined through the

FTRGetParam call with the FTR_PARAM_IMAGE_SIZE value of the first argument.

Return values

If the function succeeds, it returns the FTR_RETCODE_OK code. Otherwise, the returned value indicates an error.

Error code	Description
FTR_RETCODE_INVALID_ARG	Some parameters were not specified or had invalid values.
FTR_RETCODE_FRAME_SOURCE_NOT_SET	Attributes of the frame image become available only after setting the frame source.
FTR_RETCODE_CANCELED_BY_USER	User through the established callback function canceled operation.
FTR_RETCODE_INTERNAL_ERROR	Internal SDK or Win32 API system error.
FTR_RETCODE_DEVICE_NOT_CONNECTED	The frame source device is not connected.
FTR_RETCODE_DEVICE_FAILURE	An error on the attached scanner. The appropriate Win32 error code describing the particular error can be got by calling the <FTRGetParam> function with the value of the first argument set to FTR_PARAM_SYS_ERROR_CODE.

FTR_RETCODE_FAKE_SOURCE	Fake finger was detected.

Comments

A user defined callback function must be set prior to the FTRCaptureFrame usage. To establish a callback function, a caller must use the FTRSetParam function with the first parameter set to FTR_PARAM_CB_CONTROL.

If the plugged scanner device is used by another application, this function waits for either of two events comes first: the scanner becomes released or the FTR_CANCEL response fires through the established user callback function. This can produce an infinite delay if neither of these events comes.

FTREnroll, FTREnrollX

Both FTREnroll and FTREnrollX functions create the fingerprint template for the desired purpose.

```
FTRAPI_RESULT FTREnroll(
  FTR_USER_CTX UserContext,  // [in] optional value
  FTR_PURPOSE Purpose,       // [in] purpose of template building
  FTR_DATA_PTR pTemplate     // [out] pointer to a result memory buffer
);

FTRAPI_RESULT FTREnrollX(
  FTR_USER_CTX UserContext,  // [in] optional value
  FTR_PURPOSE Purpose,       // [in] purpose of template building
  FTR_DATA_PTR pTemplate,    // [out] pointer to a result memory buffer
  FTR_ENROLL_DATA_PTR pEData // [out] optional pointer to the FTR_ENROLL_DATA structure
);
```

Parameters

UserContext

[in] -optional caller-supplied value that is passed to callback functions. This value is provided for convenience in application design.

Purpose

[in] -the purpose of template building. This value designates the intended way of further template usage and can be one of the following:

Value	Meaning
FTR_PURPOSE_ENROLL	The created template is suitable for both identification and verification purpose.
FTR_PURPOSE_IDENTIFY	Corresponding template can be used only for identification as an input for the FTRSetBaseTemplate function.

pTemplate

[out] -pointer to a result memory buffer. A caller must reserve the space for this buffer. Maximum space amount can be determined through the FTRGetParam call with the FTR_PARAM_MAX_TEMPLATE_SIZE value of the first argument.

pEData

[out] -optional pointer to the FTR_ENROLL_DATA structure that receives on output additional information on the results of the enrollment process. The caller must set the dwSize member of this structure to sizeof(FTR_ENROLL_DATA) in order to identify the version of the structure being passed. If a caller does not initialize dwSize, the function fails.

Return values

If the function succeeds, it returns the FTR_RETCODE_OK code. Otherwise, the returned value indicates an error.

Error code	Description
FTR_RETCODE_INVALID_ARG	Some parameters were not specified or had invalid values.
FTR_RETCODE_FRAME_SOURCE_NOT_SET	Attributes of the frame image become available only after setting the frame source.
FTR_RETCODE_CANCELED_BY_USER	User through the established callback function canceled operation.
FTR_RETCODE_INTERNAL_ERROR	Internal SDK or Win32 API system error.
FTR_RETCODE_DEVICE_NOT_CONNECTED	The frame source device is not connected.
FTR_RETCODE_DEVICE_FAILURE	An error on the attached scanner. The appropriate Win32 error code describing the particular error can be got by calling the FTRGetParam function with the value of the first argument set to

	FTR_PARAM_SYS_ERROR_CODE.
FTR_RETCODE_FAKE_SOURCE	Fake finger was detected.

Comments

A user defined callback function must be set prior to the FTREnroll usage. To establish a callback function, a caller must use the FTRSetParam function with the first parameter set to FTR_PARAM_CB_CONTROL.

If the plugged scanner device is used by another application, this function waits for either of two events comes first: the scanner becomes released or the FTR_CANCEL response fires through the established user callback function. This can produce an infinite delay if neither of these events comes.

FTRGetParam

The FTRGetParam function gets the value of the specified parameter.

```
FTRAPI_RESULT FTRGetParam(
  FTR_PARAM Param,        // [in] requested parameter
  FTR_PARAM_VALUE *pValue // [out] parameter value
);
```

Parameters

Param

[in] indicates the parameter which value must be obtained. Supported parameters are described in the following table. The Type column specifies the type of data addressed by *pValue*.

Value	Type	Meaning
FTR_PARAM_IMAGE_WIDTH	DWORD	Width of the frame image on the attached device measured in pixels.
FTR_PARAM_IMAGE_HEIGHT	DWORD	Height of the frame image on the attached device measured in pixels.
FTR_PARAM_IMAGE_SIZE	DWORD	Size of the frame image in bytes.
FTR_PARAM_CB_FRAME_SOURCE	DWORD	Type of the frame source.

FTR_PARAM_CB_CONTROL	FTR_CB_STATE_CONTROL	Caller-supplied callback function used for interaction with a user during enrollment or verification.
FTR_PARAM_MAX_TEMPLATE_SIZE	DWORD	Maximum template size in bytes.
FTR_PARAM_MAX_FAR_REQUESTED	FTR_FAR	FAR level used for verification and/or identification.
FTR_PARAM_MAX_FARN_REQUESTED	FTR_FARN	FAR level used for verification and/or identification.
FTR_PARAM_SYS_ERROR_CODE	DWORD	An error code returned by the Win32 GetLastError() function in the case of the device failure. Use this value if you've got the FTR_RETCODE_DEVICE_FAILURE code upon a FTR API call completion.
FTR_PARAM_FAKE_DETECT	BOOL	Operating mode of device. Determines whether a fake finger detection mechanism is activated.

FTR_PARAM_FFD_CONTROL FTR_PARAM_MAX_MODELS FTR_PARAM_MIOT_CONTROL FTR_PARAM_VERSION	BOOL DWORD BOOL DWORD	Indicates whether a calling application has taken a control over the Fake Finger Detection (FFD) event. If this value is set to TRUE an application receives notification on any FFD event through the caller-supplied callback function. Maximum number of frames in a template that is suitable both for verification and identification purpose. Indicates if the Multifingers In One Template (MIOT) feature is enabled. With this value set to TRUE different fingers cannot be combined in the same template during the enrollment process. Version number compatibility. Returns one of the following values: FTR_VERSION_PREVIOUS, FTR_VERSION_COMPATIBLE , FTR_VERSION_CURRENT. See the FTRSetParam function for more details.

pValue

[out] pointer to a variable which receives the requested parameter value.

Return values

If the function succeeds, it returns the FTR_RETCODE_OK code. Otherwise, the returned value indicates an error.

Error code	Description
FTR_RETCODE_INVALID_ARG	Some parameters were not specified or had invalid values.

FTR_RETCODE_FRAME_SOURCE_NOT_SET	Attributes of the frame image become availabsetting the frame source.	le only after

Comments

Either of the two values can be returned if a caller issues a frame source request:

• FSD__USB -a USB Fingerprint Scanner Device has been set as a frame source.

• FSD_UNDEFINED -a frame source has not been set.

FTRIdentify

The FTRIdentify function compares the base template against a set of source templates. The matching is performed in terms of FAR (False Accepting Ratio), which designates the probability of falsely matching of the base template to the source template.

```
FTRAPI_RESULT FTRIdentify(
  FTR_IDENTIFY_ARRAY_PTR pAIdent, // [in] pointer to the set of source templates
  DWORD *pdwMatchCnt,             // [in,out] number of matched records
  FTR_MATCHED_ARRAY_PTR pAMatch   // [in,out] pointer to the array of matched records
);
```

```
FTRAPI_RESULT FTRIdentifyN(
  FTR_IDENTIFY_ARRAY_PTR pAIdent,  // [in] pointer to the set of source templates
  DWORD *pdwMatchCnt,              // [in,out] number of matched records
  FTR_MATCHED_X_ARRAY_PTR pAMatch  // [in,out] pointer to the array of matched records
);
```

Parameters

pAIdent

[in] -points to a set of the source templates.

pdwMatchCnt

[in,out] -pointer to a number of matched records in the array pointed to by the pAMatch argument. Before entering the identification loop the number of matched records must be initialized to 0.

pAMatch

[in,out] -pointer to the array of matched records. A caller is responsible for reserving appropriate memory space and proper initialization of this structure.

Return values

If the function succeeds, it returns the FTR_RETCODE_OK code. Otherwise, the returned value indicates an error.

Error code	Description
FTR_RETCODE_INVALID_ARG	Some parameters were not specified or had invalid values.
FTR_RETCODE_INVALID_PURPOSE	There is a template built with the purpose other than FTR_PURPOSE_ENROLL value in the *pAIdent* array.

Comments

Note, that in the successful completion the value pointed to by the *pdwMatchCnt* argument contains the number of matching templates, i.e. if this value is set to 0, there were no matching source templates detected, otherwise the most probable results are represented in the *pAMatch* array ordered descending by their probability.

The matching is performed according to the current FAR level, that can be set via the FTRSetParam call with the value of the first argument set either to the FTR_PARAM_MAX_FAR_REQUESTED or to the FTR_PARAM_MAX_FARN_REQUESTED value.

FTRInitialize

The FTRInitialize function activates the SDK interface. This function must be called before any other API call.

```
FTRAPI_RESULT FTRInitialize( void );
```

Return values

If the function succeeds, it returns the FTR_RETCODE_OK code. Otherwise, the returned value indicates an error.

Error code	Description	
FTR_RETCODE_ALREADY_IN_USE	The current process has already	initialized the API.
FTR_RETCODE_NO_MEMORY	Not enough memory to perform t	he operation.

FTRSetBaseTemplate

The FTRSetBaseTemplate function installs a template as a base for identification process. The passed template must have been enrolled for identification purpose, i.e. the FTR_PURPOSE_IDENTIFY purpose value must be used for its enrollment. Identification process is organized in one or more FTRIdentify calls.

```
FTRAPI_RESULT FTRSetBaseTemplate(
  FTR_DATA_PTR pTemplate  // [in] pointer to a previously enrolled template
);
```

Parameters

pTemplate

[in] -pointer to a previously enrolled template.

Return values

If the function succeeds, it returns the FTR_RETCODE_OK code. Otherwise, the returned value indicates an error.

Error code	Description
FTR_RETCODE_INVALID_ARG	Some parameters were not specified or had invalid values.
FTR_RETCODE_INVALID_PURPOSE	The input template was not built with the FTR_PURPOSE_IDENTIFY purpose.

FTRSetParam

The FTRSetParam function sets the indicated parameter value.

```
FTRAPI_RESULT FTRSetParam(
  FTR_PARAM Param,      // [in] parameter to be set
  FTR_PARAM_VALUE Value // [in] parameter value
);
```

Parameters

Param

[in] indicates the parameter which value must be set. Supported parameters are described in the following table. The Type column specifies the type of data passed in the *Value* argument.

Value	Type	Meaning

FTR_PARAM_CB_FRAME_SOURCE	DWORD	Type of the frame source.
FTR_PARAM_CB_CONTROL	FTR_CB_STATE_CONTROL	Caller-supplied callback function used for interaction with a user during enrollment or verification.
FTR_PARAM_MAX_FAR_REQUESTED	FTR_FAR	FAR level used for verification and/or identification.
FTR_PARAM_MAX_FARN_REQUESTED	FTR_FARN	FAR level used for verification and/or identification.
FTR_PARAM_FAKE_DETECT	BOOL	Operating mode of device. Determines whether a fake finger detection mechanism is activated.
FTR_PARAM_FFD_CONTROL	BOOL	Indicates whether a calling application takes a control over the Fake Finger Detection (FFD) event. If this value is set to TRUE an application receives notification on any FFD event through the caller-supplied callback function.
FTR_PARAM_MAX_MODELS	DWORD	Maximum number of frames in a template that is suitable both for verification and identification purpose. This value can vary from 3 to 10 and is equal to 7 by default.
FTR_PARAM_MIOT_CONTROL	BOOL	Indicates if the Multifingers In One Template (MIOT) feature is enabled. With this value set to TRUE different fingers cannot be combined in the same template during the enrollment process.
FTR_PARAM_VERSION	DWORD	This parameter denotes the mode of SDK functioning and can be evaluated as: -FTR_VERSION_CURRENT -the SDK uses the current version of algorithm with better statistic results. This value should be used for the new fingerprint data bases creation; -FTR_VERSION_PREVIOUS -the

		algorithm from previous SDK version 3.0 is selected. This value can be advised to clients, which have fingerprint data bases already created and they are totally satisfied with using of SDK 3.0; -FTR_VERSION_COMPATIBLE - the combined version, which allows a gradual update to the current version with better statistic results. This value is selected by default during SDK 3.6 initialization.

Value

[in] the value of the specified parameter.

Return values

If the function succeeds, it returns the FTR_RETCODE_OK code. Otherwise, the returned value indicates an error.

Error code	Description	
FTR_RETCODE_INVALID_ARG	Some parameters were not specified or had values.	invalid
FTR_RETCODE_NO_MEMORY	Not enough memory to perform the operation.	

Comments

To set a frame source use the FSD__USB value, which requires that a USB Fingerprint Scanner Device must be plugged in any available USB port. To clear the specified frame source if that has been previously set, use the FSD_UNDEFINED value.

The maximum template size value depends on a maximum number of frames in a template. This means that if a caller employs his/her own maximum number of frames he/she must invoke the FTRSetParam(FTR_PARAM_MAX_MODELS, …) function before the FTRGetParam(FTR_PARAM_MAX_TEMPLATE_SIZE, …) call.

FTRTerminate

The FTRInitialize releases all previously allocated resources and completes the API usage. This call must be the last API call in the case of SUCCESSFULL FTRInitialize return.

```
void FTRTerminate( void );
```

FTRVerify

The FTRVerify function captures an image from the currently attached frame source, builds the corresponding template and compares it with the source template passed in the *pTemplate* parameter.

```
FTRAPI_RESULT FTRVerify(
  FTR_USER_CTX UserContext,  // [in] optional value
  FTR_DATA_PTR pTemplate,    // [in] pointer to a source template
  BOOL *pResult,             // [out] points to the result of verification
  FTR_FAR *pFARVerify        // [out] optional FAR level achieved
);

FTRAPI_RESULT FTRVerifyN(
  FTR_USER_CTX UserContext,  // [in] optional value
  FTR_DATA_PTR pTemplate,    // [in] pointer to a source template
  BOOL *pResult,             // [out] points to the result of verification
  FTR_FARN *pFARVerify       // [out] optional FAR level achieved
);
```

Parameters

UserContext

[in] -optional caller-supplied value that is passed to callback functions. This value is provided for convenience in application design.

pTemplate

[in] -pointer to a source template for verification.

pResult

[out] -points to a value indicating whether the captured image matched to the source template.

pFARVerify

[out] -points to the optional FAR level achieved.

Return values

If the function succeeds, it returns the FTR_RETCODE_OK code. Otherwise, the returned value indicates an error.

Error code	Description
FTR_RETCODE_INVALID_ARG	Some parameters were not specified or had invalid values.
FTR_RETCODE_INVALID_PURPOSE	The input template was not built with the FTR_PURPOSE_ENROLL purpose.
FTR_RETCODE_FRAME_SOURCE_NOT_SET	Attributes of the frame image become available only after setting the frame source.
FTR_RETCODE_CANCELED_BY_USER	User through the established callback function canceled operation.
FTR_RETCODE_INTERNAL_ERROR	Internal SDK or Win32 API system error.
FTR_RETCODE_DEVICE_NOT_CONNECTED	The frame source device is not connected.
FTR_RETCODE_DEVICE_FAILURE	An error on the attached scanner. The appropriate Win32 error code describing the particular error can be got by calling the FTRGetParam function with the value of the first argument set to FTR_PARAM_SYS_ERROR_CODE.
FTR_RETCODE_FAKE_SOURCE	Fake finger was detected.

Comments

A user defined callback function must be set prior to the FTRVerify usage. To establish a callback function, a caller must use the FTRSetParam function with the first parameter set to FTR_PARAM_CB_CONTROL.

If the plugged scanner device is used by another application, this function waits for either of two events comes first: the scanner becomes released or the FTR_CANCEL response fires through the established user callback function. This can produce an infinite delay if neither of these events comes.

The matching is performed according to the current FAR level, that can be set via the FTRSetParam call with the value of the first argument set either to the FTR_PARAM_MAX_FAR_REQUESTED or to

the FTR_PARAM_MAX_FARN_REQUESTED value.

SDK structures definitions

FTR_ENROLL_DATA

This structure presents features of the successfully created template.

```
typedef struct {
  DWORD dwSize;
  DWORD dwQuality;
} FTR_ENROLL_DATA;
```

Members

dwSize

Specifies the size of the structure in bytes.

dwQuality

Estimation of a template quality in terms of recognition scale: the lowest value 1 corresponds to the worst quality, 10 denotes the best quality.

Comments

A calling application can access the template quality through the FTREnrollX function.

See also

FTREnrollX

FTR_PROGRESS

This structure holds the frame capture progress information.

```
typedef struct {
  DWORD dwSize;
  DWORD dwCount;
  BOOL  bIsRepeated;
  DWORD dwTotal;
} FTR_PROGRESS;
```

Members

dwSize

Specifies the size of the structure in bytes.

dwCount

Currently requested frame number.

bIsRepeated

Flag indicating whether the frame is requested not the first time.

dwTotal

Total number of frames to be captured.

Comments

A calling application receives a pointer to this structure through the state callback function.