ftrScanOpenDevice

The **ftrScanOpenDevice** function opens device on the default interface.

FTRHANDLE ftrScanOpenDevice();

Return Values

If the function succeeds, the return value is an open handle to a device.

If the function fails, the return value is **NULL**. To get extended error information, call **GetLastError**.

Remarks

Use the **ftrScanCloseDevice** function to close a device handle that **ftrScanOpenDevice** returns.
You can change the default interface number by calling **ftrSetBaseInterface** function.

Example Code

The following code example shows you how to receive frame from device.

```
FTRHANDLE hDevice;
PVOID pBuffer;
FTRSCAN_IMAGE_SIZE ImageSize;

hDevice = ftrScanOpenDevice();
if( hDevice == NULL )
{
      // process error
      // ...
} else {

      ftrScanGetImageSize( hDevice, &ImageSize );
      pBuffer = new BYTE [ImageSize.nImageSize];

      if( ftrScanGetFrame( hDevice, pBuffer, NULL ) )
      {
            // success
            // ...
      } else {
            switch( GetLastError() )
            {
            case FTR_ERROR_EMPTY_FRAME:
                  // fingerprint is not present
                  // ...
                  break;
            case FTR_ERROR_MOVABLE_FINGER:
                  // Finger is movable. Repeat the scan process.
                  // ...
                  break;
            case FTR_ERROR_NO_FRAME:
                  // "Fake finger" detected
                  // ...
```

```
                    break;
                default:
                    // process error
                    // ...
                }
        }

        delete pBuffer;
        ftrScanCloseDevice( hDevice );
}
```

ftrScanCloseDevice

The **ftrScanCloseDevice** function closes an open device handle.

```
void ftrScanCloseDevice(
        FTRHANDLE ftrHandle
        );
```

Parameters

ftrHandle

[in] Handle to an open device.

Remarks

To use operating system resources efficiently, an application should close device handles when they are no longer needed by using the **ftrScanCloseDevice** function.

ftrScanGetInterfaces

The **ftrScanGetInterfaces** function returns the device status for each interface.

```
BOOL ftrScanGetInterfaces(
        PFTRSCAN_INTERFACES_LIST pInterfaceList
        );
```

Parameters

pInterfaceList

[out] A pointer to the FTRSCAN_INTERFACES_LIST structure.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks
If the function succeeds, each element of the *InterfaceStatus* array contains one of the following values:

    FTRSCAN_INTERFACE_STATUS_CONNECTED
    FTRSCAN_INTERFACE_STATUS_DISCONNECTED

The maximum number of devices is 128.
You can open any connected device by calling **ftrScanOpenDeviceOnInterface** function or you can change the default interface number by calling **ftrSetBaseInterface**.

ftrSetBaseInterface

The **ftrSetBaseInterface** function sets the new default interface number.

```
BOOL ftrSetBaseInterface(
        int nBaseInterface
        );
```

Parameters

ftrHandle
: [in] The new default interface number.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

ftrGetBaseInterfaceNumber

The **ftrGetBaseInterfaceNumber** function returns the default interface number.

```
int ftrGetBaseInterfaceNumber();
```

Return Values

The **ftrGetBaseInterfaceNumber** function returns the default interface number. The return value must be between 0 and 127.

ftrScanOpenDeviceOnInterface

The **ftrScanOpenDeviceOnInterface** function opens device on the selected interface.

```
FTRHANDLE ftrScanOpenDeviceOnInterface( int nInterface );
```

Parameters

nInterface

[in] Index of the device.

Return Values

If the function succeeds, the return value is an open handle to a device.

If the function fails, the return value is **NULL**. To get extended error information, call **GetLastError**.

Remarks

You can find the connected devices by calling the **ftrScanGetInterfaces** function.

Use the **ftrScanCloseDevice** function to close a device handle that **ftrScanOpenDeviceOnInterface** returns.

ftrScanSetOptions

The **ftrScanSetOptions** function changes the handle specific options.

```
BOOL ftrScanSetOptions(
	FTRHANDLE ftrHandle,
	DWORD dwMask,
	DWORD dwFlags
	);
```

Parameters

ftrHandle

[in] A handle to the device.

dwMask

[in] Mask that specifies the bit flags to be changed. Use the same constants shown in the description of *dwFlags*.

dwFlags

[in] Set of bit flags that specifies properties of the device handle. This parameter can be one of the following values.

Value	Meaning
FTR_OPTIONS_DETECT_FAKE_FINGER	If this flag is set, calling the **ftrScanGetFrame** function will activate Live Finger Detection (LFD) feature during the capture process.
FTR_OPTIONS_FAST_FINGER_DETECT_METHOD	If this flag is set, calling the **ftrScanIsFingerPresent** function will return FTR_ERROR_EMPTY_FRAME much faster, if no finger is available.
FTR_OPTIONS_RECEIVE_LONG_IMAGE	If this flag is set, the height of the image is doubled, if it's supported by device. This flag cannot be used with FTR_OPTIONS_IMPROVE_IMAGE and FTR_OPTIONS_SCALE_IMAGE flags.
FTR_OPTIONS_RECEIVE_FAKE_IMAGE	If this flag is set, calling the **ftrScanGetFrame** function will capture a fingerprint image, even when **ftrScanGetFrame** returns FTR_ERROR_NO_FRAME.
FTR_OPTIONS_SCALE_IMAGE	If this flag is set, the output image will be scaled to the USB 1.1 device type geometry. This flag is ignored for USB 1.1 device type and cannot be used with the FTR_OPTIONS_RECEIVE_LONG_IMAGE flag.
FTR_OPTIONS_IMPROVE_IMAGE	If this flag is set, the output image will be improved to be compliant with PIV-071006 Image Quality Specification, if it's supported by device. This flag is set by default for PIV-compatible devices. This flag cannot be used with the FTR_OPTIONS_RECEIVE_LONG_IMAGE flag.
FTR_OPTIONS_INVERT_IMAGE	If this flag is set, the output image will be inverted.
FTR_OPTIONS_PREVIEW_MODE	If this flag set, the output image will be smaller(or same) size for speed up capture/download. FS60 only.
FTR_OPTIONS_IMAGE_FORMAT_MASK FTR_OPTIONS_IMAGE_FORMAT_1	This options setup output image format(3.2x3.0" or 1.6x1.5") for FS60.
FTR_OPTIONS_ELIMINATE_BACKGROUND	By enable this option API scan and store background. All next frames will be without background. Time for options set ~2 scan frames. Please request remove fingers from prism before options is set. Required for every used scan

	mode/preview mode.
FTR_OPTIONS_IMPROVE_BACKGROUND	This option extends dynamic range of the fingerprint image or/and improves the contrast of fingerprint image for FS90B and FS98.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

The capture time is increasing, when you activate the LFD feature.

Set the FTR_OPTIONS_FAST_FINGER_DETECT_METHOD flag only if the time of the finger unavailability is critical.

If you change the FTR_OPTIONS_RECEIVE_LONG_IMAGE flag, the image size and pixel size may be changed. Use the **ftrScanGetImageSize** function to retrieve the new image size and The **ftrScanGetDeviceInfo** function to retrieve the device specific information.

Example Code

The following code example shows you how to activate the LFD feature.

```
FTRHANDLE hDevice;

hDevice = ftrScanOpenDevice();
if( hDevice == NULL )
{
      // process error
      // ...
} else {

      if( ftrScanSetOptions( hDevice,
            FTR_OPTIONS_CHECK_FAKE_REPLICA,
            FTR_OPTIONS_CHECK_FAKE_REPLICA ) )
      {
            // success
            // ...
      } else {
            // process error
            // ...
      }

      ftrScanCloseDevice( hDevice );
}
```

ftrScanGetOptions

The **ftrScanGetOptions** function retrieves the handle specific options.

```
BOOL ftrScanGetOptions(
        FTRHANDLE ftrHandle,
        LPDWORD lpdwFlags
        );
```

Parameters

ftrHandle

[in] A handle to the device.

lpdwFlags

[out] Pointer to a variable that receives a set of bit flags that specify properties of the device handle. Use the same constants shown in the description of *dwFlags* of the **ftrScanSetOptions** function.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Example Code

The following code example shows you how to activate the LFD feature.

```
FTRHANDLE hDevice;

hDevice = ftrScanOpenDevice();
if( hDevice == NULL )
{
      // process error
      // ...
} else {

      if( ftrScanSetOptions( hDevice,
            FTR_OPTIONS_CHECK_FAKE_REPLICA,
            FTR_OPTIONS_CHECK_FAKE_REPLICA ) )
      {
            // success
            // ...
      } else {
            // process error
            // ...
      }

      ftrScanCloseDevice( hDevice );
```

```
        }
```

ftrScanGetDeviceInfo

The **ftrScanGetDeviceInfo** function retrieves the device specific information.

```
BOOL ftrScanGetDeviceInfo(
        FTRHANDLE ftrHandle,
        PFTRSCAN_DEVICE_INFO pDeviceInfo
        );
```

Parameters

ftrHandle

[in] A handle to the device.

pDeviceInfo

[in/out] A pointer to the FTRSCAN_DEVICE_INFO structure.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

For **ftrScanGetDeviceInfo** to work properly, the size of the FTRSCAN_DEVICE_INFO structure must be stored in the **dwStructSize** member of that structure.

If the function succeeds, the following information from the *pDeviceInfo* structure is available.

Value	Meaning
byDeviceCompatibility	0 – USB 1.1 device, 1 – USB 2.0 device (SOI966 based) 4 – USB 2.0 device (SOI968 based) 7 - FS50 compatible 8 – FS60 compatible
wPixelSizeX	Width of pixel in uM
wPixelSizeY	Height of pixel in uM

Example Code
The following code example shows you how to retrieve the device specific information.

```
FTRHANDLE hDevice;

hDevice = ftrScanOpenDevice();
if( hDevice == NULL )
{
      // process error
      // ...
} else {
      FTRSCAN_DEVICE_INFO DevInfo;

      // Initialize the FTRSCAN_DEVICE_INFO structure.
      ZeroMemory( &DevInfo, sizeof(DevInfo) );
      DevInfo.dwStructSize = sizeof( DevInfo );

      if(ftrScanGetDeviceInfo( hDevice, &DevInfo ) )
      {
            // success, device info is available
            // ...
      } else {
            // process error
            // ...
      }

      ftrScanCloseDevice( hDevice );
}
```

ftrScanGetImageSize

The **ftrScanGetImageSize** function retrieves the image size.

```
BOOL ftrScanGetImageSize(
      FTRHANDLE ftrHandle,
      PFTRSCAN_IMAGE_SIZE pImageSize
      );
```

Parameters

ftrHandle
[in] A handle to the device.

pDeviceInfo
[out] A pointer to the FTRSCAN_IMAGE_SIZE structure.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

ftrScanGetImage

The **ftrScanGetImage** function captures a raw image from the device.

> **Note:** This function is provided only for compatibility with previous versions of the ftrScanAPI.dll. New applications should use the **ftrScanGetImage2** function.

```
BOOL ftrScanGetImage(
        FTRHANDLE ftrHandle,
        int nDose,
        PVOID pBuffer
        );
```

Parameters

ftrHandle

[in] A handle to the device.

nDose

[in] Durability of infrared led. The value must be in the **1 – 4** range.

pBuffer

[out] A pointer to the buffer that receives the raw image.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

You can use the **ftrScanGetImageSize** function to retrieve the minimum buffer size.

ftrScanGetImage2

The **ftrScanGetImage2** function captures a raw image from the device.

```
BOOL ftrScanGetImage2(
        FTRHANDLE ftrHandle,
        int nDose,
        PVOID pBuffer
        );
```

Parameters

ftrHandle
[in] A handle to the device.

nDose
[in] Durability of infrared led. The value must be in the **1 – 7** range.

pBuffer
[out] A pointer to the buffer that receives the raw image.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

You can use the **ftrScanGetImageSize** function to retrieve the minimum buffer size.

ftrScanGetDarkImage

The **ftrScanGetImage** function captures a raw image from the device without any internal illumination.

```
BOOL ftrScanGetDarkImage(
        FTRHANDLE ftrHandle,
        PVOID pBuffer
        );
```

Parameters

ftrHandle
[in] A handle to the device.

pBuffer

[out] A pointer to the buffer that receives the raw image.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

You can use the **ftrScanGetImageSize** function to retrieve the minimum buffer size.

ftrScanIsFingerPresent

The **ftrScanIsFingerPresent** function returns a Boolean value that indicates whether the fingerprint is present on the device.

```
BOOL ftrScanIsFingerPresent(
        FTRHANDLE ftrHandle,
        PFTRSCAN_FRAME_PARAMETERS pFrameParameters
        );
```

Parameters

ftrHandle

[in] A handle to the device.

pFrameParameters

[in] This parameter is reserved and must be set to NULL.

Return Values

If the fingerprint presents, the return value is **TRUE**.

If the function fails or fingerprint is not presents, the return value is **FALSE**.

To get extended error information, call **GetLastError**.

ftrScanGetFrame

The **ftrScanGetFrame** function captures a frame from the device.

```
BOOL ftrScanGetFrame(
        FTRHANDLE ftrHandle,
        PVOID pBuffer,
        PFTRSCAN_FRAME_PARAMETERS pFrameParameters );
```

Parameters

ftrHandle
[in] A handle to the device.

pBuffer
[out] A pointer to the buffer that receives the frame.

pFrameParameters
[in] This parameter is reserved and must be set to NULL.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

This function is the main function to receive the fingerprint image from the device. Use this function when you need to capture the most appropriate image.

You can use the **ftrScanGetImageSize** function to retrieve the minimum buffer size.

To activate the Live Finger Detection (LFD) feature use the **ftrScanSetOptions** function.

ftrScanSave7Bytes

The **ftrScanSave7Bytes** function stores a 7-bytes length buffer on the device.

```
BOOL ftrScanSave7Bytes(
        FTRHANDLE ftrHandle,
        PVOID pBuffer
        );
```

Parameters

ftrHandle

[in] A handle to the device.

pBuffer

[in] A pointer to the 7-bytes length buffer.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

The **ftrScanSave7Bytes** function stores a user buffer on the device. Any application may read this buffer by calling the **ftrScanRestore7Bytes** function.

ftrScanRestore7Bytes

The **ftrScanRestore7Bytes** function restores a 7-bytes length buffer from the device.

```
BOOL ftrScanRestore7Bytes(
	FTRHANDLE ftrHandle,
	PVOID pBuffer
	);
```

Parameters

ftrHandle

[in] A handle to the device.

pBuffer

[out] A pointer to the buffer that receives 7 bytes from the device.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks
Length of the buffer must be at least 7 bytes long.

ftrScanSetNewAuthorizationCode

The **ftrScanSetNewAuthorizationCode** stores the authorization code to use with **ftrScanSaveSecret7Bytes**/**ftrScanRestoreSecret7Bytes** functions.

```
BOOL ftrScanSetNewAuthorizationCode(
        FTRHANDLE ftrHandle,
        PVOID pSevenBytesAuthorizationCode
        );
```

Parameters

ftrHandle
    [in] A handle to the device.

pSevenBytesAuthorizationCode
    [in] A pointer to the 7-bytes length authorization code.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

Length of the authorization code must be 7 bytes long.

The stored authorization code **CAN NOT** be changed.

ftrScanSaveSecret7Bytes

The **ftrScanSave7Bytes** function stores a 7-bytes length buffer on the device.

```
BOOL ftrScanSaveSecret7Bytes(
        FTRHANDLE ftrHandle,
        PVOID pSevenBytesAuthorizationCode,
        PVOID pBuffer
        );
```

Parameters

ftrHandle
[in] A handle to the device.

pSevenBytesAuthorizationCode
[in] A pointer to the 7-bytes length authorization code.

pBuffer
[in] A pointer to the 7-bytes length buffer.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**. If the authorization code wrong is, **GetLastError** returns **FTR_ERROR_INVALID_AUTHORIZATION_CODE.**

Remarks
The **ftrScanSaveSecret7Bytes** function stores a user buffer on the device. You can read this buffer by calling the **ftrScanRestoreSecret7Bytes** function.

ftrScanRestoreSecret7Bytes

The **ftrScanRestoreSecret7Bytes** function restores a 7-bytes length buffer from the device.

```
BOOL ftrScanRestoreSecret7Bytes(
        FTRHANDLE ftrHandle,
        PVOID pSevenBytesAuthorizationCode,
        PVOID pBuffer
        );
```

Parameters

ftrHandle
[in] A handle to the device.

pSevenBytesAuthorizationCode
[in] A pointer to the 7-bytes length authorization code.

pBuffer
[out] A pointer to the buffer that receives 7 bytes from the device.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**. If the authorization code is wrong, **GetLastError** returns **FTR_ERROR_INVALID_AUTHORIZATION_CODE.**

Remarks
Length of the buffer must be at least 7 bytes long.

ftrScanSetDiodesStatus

The **ftrScanSetDiodesStatus** sets new status to green and red gimmick diodes.

```
BOOL ftrScanSetDiodesStatus(
        FTRHANDLE ftrHandle,
        BYTE byGreenDiodeStatus,
        BYTE byRedDiodeStatus
        );
```

Parameters

ftrHandle
[in] A handle to the device.

byGreenDiodeStatus
[in] New status for the green gimmick diode

byRedDiodeStatus
[in] New status for the red gimmick diode

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks
Red and green statuses can be one of the following values.

Value	Meaning
0	Turn off gimmick diode
1 - 254	Turn on gimmick diode for the specific time period in 10 msec units

255	Turn on gimmick diode

ftrScanGetDiodesStatus

The **ftrScanGetDiodesStatus** returns current status of green and red gimmick diodes.

```
BOOL ftrScanGetDiodesStatus(
        FTRHANDLE ftrHandle,
        PBOOL pbIsGreenDiodeOn,
        PBOOL pbIsRedDiodeOn
        );
```

Parameters

ftrHandle

[in] A handle to the device.

pbIsGreenDiodeOn

[in] Current status of the green gimmick diode

pbIsRedDiodeOn

[in] Current status of the red gimmick diode

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

ftrScanGetExtMemorySize

The **ftrScanGetExtMemorySize** function retrieves the EEPROM size in bytes.

```
BOOL ftrScanGetExtMemorySize(
        FTRHANDLE ftrHandle,
        int *pnSize
        );
```

Parameters

ftrHandle

[in] A handle to the device.

pnSize
[out] A pointer to the variable that receives the EEPROM size.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

ftrScanSaveExtMemory

The **ftrScanSaveExtMemory** function stores data to EEPROM at the specified position.

```
BOOL ftrScanSaveExtMemory(
        FTRHANDLE ftrHandle,
        FTR_PVOID pBuffer,
        int nOffset,
        int nCount );
```

Parameters

ftrHandle
[in] A handle to the device.

pBuffer
[in] A pointer to the buffer containing the data to be written to EEPROM.

nOffset
[in] Specifies the number of bytes to offset the EEPROM pointer.

nCount
[in] Number of bytes to be written to EEPROM.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks
The **ftrScanSaveExtMemory** function stores a user buffer to EEPROM on the device. Any application may read this buffer by calling the **ftrScanRestoreExtMemory** function.

ftrScanRestoreExtMemory

The **ftrScanRestoreExtMemory** function restores data from EEPROM at the specified position.

```
BOOL ftrScanRestoreExtMemory(
        FTRHANDLE ftrHandle,
        PVOID pBuffer,
        int nOffset,
        int nCount
        );
```

Parameters

ftrHandle
[in] A handle to the device.

pBuffer
[out] A pointer to the buffer that receives the data read from EEPROM.

nOffset
[in] Specifies the number of bytes to offset the EEPROM pointer.

nCount
[in] Number of bytes to be read from EEPROM.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks
Length of the buffer must be at least nCount bytes long.

ftrScanGetSerialNumber

The **ftrScanGetSerialNumber** function returns the 8-bytes length Serial Number of the device.

```
BOOL ftrScanGetSerialNumber(
        FTRHANDLE ftrHandle,
        PVOID pBuffer
        );
```

Parameters

ftrHandle

[in] A handle to the device.

pBuffer

[out] A pointer to the buffer that receives the 8-bytes length Serial Number.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

Length of the buffer must be at least 8 bytes long.

ftrScanGetVersion

The **ftrScanGetVersion** function returns the current version number of the API, the device hardware version and the device firmware version.

```
BOOL ftrScanGetVersion(
        FTRHANDLE ftrHandle,
        PFTRSCAN_VERSION_INFO pVersionInfo
        );
```

Parameters

ftrHandle

[in] A handle to the device.

pVersionInfo

[in/out] A pointer to the FTRSCAN_VERSION_INFO structure.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

For **ftrScanGetDeviceInfo** to work properly, the size of the FTRSCAN_VERSION_INFO structure must be stored in the **dwVersionInfoSize** member of that structure.
If a handle to the device is invalid or NULL, **ftrScanGetDeviceInfo** returns only the API version.

Example Code

The following code example shows you how to receive the API version.

```
FTRSCAN_VERSION_INFO ftrVersionInfo;
ftrVersionInfo.dwVersionInfoSize = sizeof( FTRSCAN_VERSION_INFO );

if( ftrScanGetVersion( NULL, &ftrVersionInfo ) )
{
      // ftrVersionInfo.APIVersion structure contains API version
      // ...
}
```

ftrSetLoggingFacilityLevel

The **ftrSetLoggingFacilityLevel** function changes the logging facility level.

```
BOOL ftrSetLoggingFacilityLevel(
	DWORD dwLogMask,
	DWORD dwLogLevel,
	char * lpFileName
	);
```

Parameters

dwLogMask

[in] Specify where logging output should be directed.

Value	Meaning
FTR_LOG_MASK_OFF	Disable logging facility. This attribute is valid only if used alone. This is the default.
FTR_LOG_MASK_TO_FILE	Write log messages to the specified file.
FTR_LOG_MASK_TO_AUX	Send log messages to the debugger for display. If the application has no debugger, the system debugger displays the log. If the application has no debugger and the system debugger is not active, this flag is ignored.
FTR_LOG_MASK_TIMESTAMP	Add current date/time to log messages.
FTR_LOG_MASK_THREAD_ID	Add the calling thread ID to log messages.
FTR_LOG_MASK_PROCESS_ID	Add the calling process ID to log messages.

dwLogLevel

[in] Specifies the log level. This parameter can be one of the following values:

FTR_LOG_LEVEL_MIN
FTR_LOG_LEVEL_OPTIMAL
FTR_LOG_LEVEL_FULL

lpFileName

[in/optional] Specifies the name of the log file.

Return Values

If the function succeeds, the return value is **TRUE**. If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

This function greatly decreases the capturing time. Use this function only if you know what you do!

ftrScanRollStart

The **ftrScanRollStart** function starts the roll operation.

```
BOOL ftrScanRollStart (
        FTRHANDLE ftrHandle
        );
```

Parameters

ftrHandle

[in] A handle to the device.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

If this function succeeds, the roll operation is started and all other operations are blocked from any other threads and applications. The device will be unblocked only, when the final image is produced, or the roll operation is aborted.

ftrScanRollStarWithVariableDose

The **ftrScanRollStarWithVariableDose** function starts the roll operation.

```
BOOL ftrScanRollStarWithVariableDose (
        FTRHANDLE ftrHandle,
        int nVariableDose
        );
```

Parameters

ftrHandle

[in] A handle to the device.

nVariableDose

[in] Durability of infrared led. The value must be in the **0 – 255** range.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

See remarks for the ftrScanRollStart function.

ftrScanRollAbort

The **ftrScanRollAbort** function aborts the roll operation.

```
BOOL ftrScanRollAbort (
        FTRHANDLE ftrHandle,
        BOOL bSynchronous
        );
```

Parameters

ftrHandle

[in] A handle to the device.

bSynchronous

[in] If this parameter is **TRUE**, ftrScanRollAbort waits for end of roll operation.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

ftrScanRollGetImage

The **ftrScanRollGetImage** function receives a current roll image.

```
BOOL ftrScanRollGetImage (
        FTRHANDLE ftrHandle,
        PVOID pBuffer,
        DWORD dwMilliseconds
        );
```

Parameters

ftrHandle

[in] A handle to the device.

pBuffer

[out] A pointer to the buffer that receives the image.

dwMilliseconds

[in] The time-out interval, in milliseconds. The function returns if the interval elapses, even if the new image is not prepared. If dwMilliseconds is **FTR_TIMEOUT_INFINITE**, the function's time-out interval never elapses.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

You can use the **ftrScanGetImageSize** function to retrieve the minimum buffer size. If the function fails, you should check the **GetLastError** function for the **FTR_ERROR_ROLL_PROGRESS_DATA** return code. If the **GetLastError** function returns this code, the roll operation is not finished and **pBuffer** contains intermediate image.

ftrScanRollGetFrameParameters

The **ftrScanRollGetFrameParameters** function returns detailed information about the current roll process.

```
BOOL ftrScanRollGetFrameParameters(
    FTRHANDLE ftrHandle,
    PFTRSCAN_ROLL_FRAME_PARAMETERS pFrameParameters,
    FTR_PVOID pBuffer,
    FTR_DWORD dwMilliseconds
    );
```

Parameters

ftrHandle

[in] A handle to the device.

pFrameParameters
[in/out] A pointer to the **FTRSCAN_ROLL_FRAME_PARAMETERS** structure. Before calling the **ftrScanRollGetFrameParameters** function, set the **dwSize** member of the **FTRSCAN_ROLL_FRAME_PARAMETERS** data structure to **sizeof(FTRSCAN_ROLL_FRAME_PARAMETERS).**

pBuffer
[out] A pointer to the buffer that receives the image.

dwMilliseconds
[in] The time-out interval, in milliseconds. The function returns if the interval elapses, even if the new image is not prepared. If dwMilliseconds is **FTR_TIMEOUT_INFINITE**, the function's time-out interval never elapses.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Remarks

You can use the **ftrScanGetImageSize** function to retrieve the minimum buffer size for the image. If the function fails, you should check the **GetLastError** function for the **FTR_ERROR_ROLL_PROGRESS_PUT_FINGER**, **FTR_ERROR_ROLL_PROGRESS_REMOVE_FINGER**, **FTR_ERROR_ROLL_PROGRESS_POST_PROCESSING** and **FTR_ERROR_ROLL_PROGRESS_DATA** return codes. If the **GetLastError** function returns one of thess codes, the roll operation is not finished and **pFrameParameters** contains intermediate data about the roll process and **pBuffer** contains intermediate image.

ftrScanGetProperty

The **ftrScanGetProperty** function retrieves the specified device property.

```
BOOL ftrScanGetProperty(
        FTRHANDLE ftrHandle,
        int nProperty,
        FTR_PVOID pPropertyData
        );
```

Parameters

ftrHandle

[in] A handle to the device.

nProperty

[in] The device property to be retrieved.

Value	Meaning
FTR_PROPERTY_NUMBER_OF_IMAGE_SIZES	The number of image sizes that the **ftrScanGetImageSizes** returns. The pPropertyData must be a pointer to the FTRSCAN_PROPERTY_NUMBER_OF_IMAGE_SIZES structure.
FTR_PROPERTY_LFD_SW_1_CALCULATED_DATA	The calculated strength for the LFD SW1 algorithm. The pPropertyData must be a pointer to the FTRSCAN_PROPERTY_LFD_SW_1_DATA structure.
FTR_PROPERTY_LFD_SW_1_PARAM	The current selected strength parameter for the LFD SW1 algorithm. The pPropertyData must be a pointer to the FTRSCAN_PROPERTY_LFD_SW_1_PARAM structure.

pPropertyData

[out] A pointer to buffer to store the value of the device property.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

ftrScanSetProperty

The **ftrScanSetProperty** function sets the specified device property.

```
BOOL ftrScanSetProperty(
        FTRHANDLE ftrHandle,
        int nProperty,
        FTR_PVOID pPropertyData
        );
```

Parameters

ftrHandle

[in] A handle to the device.

nProperty

[in] The device property to be set.

Value	Meaning
FTR_PROPERTY_LFD_LEVEL	The new LFD level. The pPropertyData must be a pointer to the FTRSCAN_PROPERTY_LFD_LEVEL structure. The dwLfdLevel in the FTRSCAN_PROPERTY_LFD_LEVEL structure contains set of bit flags that specifies required LFD algprithms. The recommended values are FTR_LFD_LEVEL_1, FTR_LFD_LEVEL_2 etc.
FTR_PROPERTY_LFD_SW_1 _PARAM	The new strength parameter for the LFD SW1 algorithm. The pPropertyData must be a pointer to the FTRSCAN_PROPERTY_LFD_SW_1_PARAM structure.

pPropertyData

[in] A pointer to buffer to store the value of the device property.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

ftrScanGetImageSizes

The **ftrScanGetImageSizes** function retrieves the array of image sizes supported by the device.

```
BOOL ftrScanGetImageSizes (
        FTRHANDLE ftrHandle,
        PFTRSCAN_IMAGE_SIZE pArrayOfImageSizes
        );
```

Parameters

ftrHandle

[in] A handle to the device.

pArrayOfImageSizes

[out] A pointer to array of the FTRSCAN_IMAGE_SIZE structure. To determine size of array, call the **ftrScanGetProperty** function with the FTR_PROPERTY_NUMBER_OF_IMAGE_SIZES parameter.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

ftrScanGetImageOfSpecificSize

The **ftrScanGetImageOfSpecificSize** function captures a raw image with specified size from the device.

```
BOOL ftrScanGetImageOfSpecificSize (
        FTRHANDLE ftrHandle,
        int nVariableDose,
        FTR_BYTE byLights,
        int nWidth,
        int nHeight,
        FTR_PVOID pBuffer
        );
```

Parameters

ftrHandle

[in] A handle to the device.

nVariableDose

[in] Durability of infrared led. The value must be in the **0 – 255** range.

byLights
[in] Combination of the LEDs that are turned **ON** during capture.

nWidth
[in] Width of the image to be captured.

nHeight
[in] Height of the image to be captured.

pBuffer
[out] A pointer to the buffer that receives the image.

Return Values

If the function succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To get extended error information, call **GetLastError**.

Error Codes

To get extended error information, you can call **GetLastError**. To obtain an error string for system error codes, use the **FormatMessage** function.

The following example shows an error-handling function that prints the error message.

```
void ErrorPrint(LPTSTR lpszFunction)
{
     TCHAR szBuf[80];
     LPVOID lpMsgBuf;
     DWORD dw = GetLastError();

     FormatMessage(
          FORMAT_MESSAGE_ALLOCATE_BUFFER |
          FORMAT_MESSAGE_FROM_SYSTEM,
          NULL,
          dw,
          MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
          (LPTSTR) &lpMsgBuf,
          0, NULL );

     _tcprintf (szBuf, _T( "%s failed with error %d: %s" ),
          lpszFunction, dw, lpMsgBuf );

     MessageBox( NULL, szBuf, _T( "Error" ), MB_OK );
     LocalFree( lpMsgBuf );
}
```

The following error codes are defined in 'ftrScanAPI.h'. They are used by the ftrScanAPI.dll to indicate library specific errors.

Value	Meaning
FTR_ERROR_EMPTY_FRAME	Returned by ftrScanIsFingerPresent or ftrScanGetFrame to indicate there is no fingerprint on the device.
FTR_ERROR_MOVABLE_FINGER	Returned by ftrScanGetFrame only to indicate there is no stable fingerprint image on the device.
FTR_ERROR_NO_FRAME	Returned by ftrScanGetFrame only to indicate that fake finger was detected.
FTR_ERROR_USER_CANCELED	Operation canceled by user.

	Reserved for future use.
FTR_ERROR_HARDWARE_INCOMPATIBLE	The device does not support the requested feature. The most probable reason is the device does not support the LFD.
FTR_ERROR_FIRMWARE_INCOMPATIBLE	The device does not support the requested feature. You should upgrade the device firmware to support it.
FTR_ERROR_INVALID_AUTHORIZATION_CODE	Returned by ftrScanSaveSecret7Bytes or ftrScanRestoreSecret7Bytes to indicate the specified authorization code is not correct.
FTR_ERROR_ROLL_NOT_STARTED	Returned by ftrScanRollAbort or ftrScanRollGetImage to indicate the roll operation is not started.
FTR_ERROR_ROLL_ALREADY_STARTED	Returned by ftrScanRollAbort to indicate the roll operation is already started.
FTR_ERROR_ROLL_TIMEOUT	Returned by ftrScanRollGetImage and ftrScanRollGetFrameParameters to indicate the time-out interval elapsed, and no captured data available.
FTR_ERROR_ROLL_PROGRESS_DATA	Returned by ftrScanRollGetImage and ftrScanRollGetFrameParameters to indicate the intermediate captured data are available. The roll process is still continuing.
FTR_ERROR_ROLL_ABORTED	Returned by ftrScanRollGetImage and ftrScanRollGetFrameParameters to indicate the roll process was aborted using the ftrScanRollAbort function.
FTR_ERROR_ROLL_PROGRESS_PUT_FINGER	Returned by ftrScanRollGetFrameParameters to indicate the roll process waits when user puts his finger on the device. The roll process is still continuing.
FTR_ERROR_ROLL_PROGRESS_REMOVE_FINGER	Returned by ftrScanRollGetFrameParameters to indicate the roll process waits when user removes his finger from the device. The roll process

	is still continuing.
FTR_ERROR_ROLL_PROGRESS_POST_PROCESSING	Returned by ftrScanRollGetFrameParameters to indicate the all necessary data is collected and the post processing in the progress. The roll process is almost finished, but still continuing.