

DigitalPersona, Inc.

One Touch for CE SDK

Version 1.3.0

Developer Guide



DigitalPersona, Inc.

© 1996 - 2008 DigitalPersona, Inc. All Rights Reserved.

All intellectual property rights in the Digital Persona software, firmware, hardware and documentation included with or described in this Guide are owned by Digital Persona or its suppliers and are protected by United States copyright laws, other applicable copyright laws, and international treaty provisions.

Digital Persona and its suppliers retain all rights not expressly granted.

U.are.U®, DigitalPersona® and One Touch® are registered trademarks of DigitalPersona, Inc.

Windows, Windows 2000, Windows 2003, Windows XP, and Windows CE are registered trademarks of Microsoft Corporation.

ARM® is a registered trademark of ARM Limited.

XScale® is a registered trademark of Intel Corporation.

All other trademarks are property of their respective owners.

This DigitalPersona® One Touch for CE SDK Guide and the software it describes are furnished under license as set forth in the Installation Software screen "License Agreement." Except as permitted by such license, no part of this document may be reproduced, stored, transmitted and translated, in any form and by any means, without the prior written consent of DigitalPersona. The contents of this manual are furnished for informational use only and are subject to change without notice. Any mention of third-party companies and products is for demonstration purposes only and constitutes neither an endorsement nor a recommendation. DigitalPersona assumes no responsibility with regard to the performance or use of these third-party products. DigitalPersona makes every effort to ensure the accuracy of its documentation and assumes no responsibility or liability for any errors or inaccuracies that may appear in it. This document is subject to the DigitalPersona LIMITED WARRANTY and other general provisions set forth in the Appendix of this manual.

Technical Support

Upon your purchase of a Developer Support package (available from <http://buy.digitalpersona.com>), you are entitled to a specified number of hours of telephone and email support.

Feedback

Although the information in this guide has been thoroughly reviewed and tested, we welcome your feedback on any errors, omissions, or suggestions for future improvements. Please contact us at

TechPubs@digitalpersona.com

or

DigitalPersona, Inc.

720 Bay Road, Suite 100

Redwood City, California 94063

USA

(650) 474-4000

(650) 298-8313 Fax

Table of Contents

1	Introduction	1
	Overview	1
	Target Audience	2
	Chapter Overview	2
	Document Conventions	3
	Notational Conventions	3
	Typographical Conventions	3
	Naming Conventions	3
	Additional Resources	4
	Related Documentation	4
	Online Resources	4
	System Requirements	5
	Development Platform	5
	Target (CE) Platform	5
	DigitalPersona Application Compatibility	5
2	Installation & Deployment	6
	Installing the SDK	6
	Installing the Runtime	6
3	Fingerprint Device Driver Interfaces	7
	Overview	7
	IDeviceManager interface	7
	IDeviceManager methods	8
	OpenDeviceManager	8
	GetVersion	8
	CloseDeviceManager	9
	EnumerateDevices	9
	GetDeviceInfo	10
	OpenDevice	10
	CloseDevice	11
	Release	11
	IDevice interface	12
	IDevice methods	12
	GetParameter	12
	SetParameter	12
4	Fingerprint Recognition Engine Interface	13
	Overview	13
	Module Initialization, Termination and Settings	13
	FX_getVersionInfo	13
	FX_init	13
	FX_createContext	14
	FX_closeContext	14
	FX_terminate	14
	Feature Extraction	15

Table of Contents

FX_getFeaturesLen	15
FX_extractFeatures	16
FX_getDisplayImage	17
5 Matching Functions	19
Overview	19
Module Initialization, Termination and Settings	19
MC_init	19
MC_terminate	19
MC_createContext	20
MC_closeContext	20
MC_getVersionInfo	20
MC_getSettings	21
MC_setSecurityLevel	21
MC_getSecurityLevel	22
Feature registration and verification	23
MC_getFeaturesLen	23
MC_verifyFeaturesEx	24
MC_generateRegFeatures	26
6 Index	29

The DigitalPersona One Touch for CE Software Development Kit (SDK) can be used to develop a wide variety of custom applications related to Personal Digital Assistants (PDAs), client/server access, time and attendance, and physical access. A system integrator or application developer can use the SDK to easily add the functionality of authenticating a user with fingerprint recognition to other software.

This SDK supports X86 processors running CE.NET version 6.0, and is designed specifically to utilize the Digital Persona 4000 B fingerprint readers, models R100 and R101.

DigitalPersona One Touch for CE SDK templates are interoperable with those generated by other DigitalPersona SDKs. However, matching templates coming from different digital Persona SDKs may result in a small reduction in matching accuracy.

The SDK contains this documentation, header files that define the API, the Fingerprint Recognition libraries, the fingerprint reader server, and Visual C++ sample code.

Overview

The DigitalPersona One Touch for CE SDK provides the application developer with two levels of APIs:

- Device Manager and Device functions
- Low-level Feature Extraction and Matching functions

Features, in the context of this document, are mathematical representations extracted from an image of a fingerprint that are used to describe the unique characteristics of the fingerprint. Feature extraction is the process of extracting the features (called a template) from the scan acquired from the fingerprint reader. The feature extraction process takes approximately 1 second on a 400 MHz XScale - based PDA. The feature extraction module also returns diagnostic information about the quality of the fingerprint captured, such as poor contrast, or blurred image.

The matching module performs registration and matching. Registration produces a registration template by processing 4 pre-registration features. Matching compares a registration template and verification features, and verifies that the fingerprints come from the same finger. The match process takes approximately 0.8 sec. on a 400 MHz XScale - based PDA.

The application can set the security level (False Accept Rate) to be used for matches. The default security setting are: 1.1% chance of false reject versus 0.001% chance of false accept for eXtended Template Format (XTF) and 3.5% chance of false reject for Basic Template Format (BTF).

The matching module is also able to perform learning upon successful match. Learning requires extra computation and can be switched on and off by the application.

Target Audience

This guide is for developers who have a working knowledge of the Windows CE.Net programming language. In addition, some familiarity with fingerprint recognition and identification concepts and processes is recommended.

The DigitalPersona White Paper: Guide to Fingerprint Recognition (Fingerprint Guide.pdf), located in the Docs folder of the product package, can provide a basic introduction to these concepts.

Chapter Overview

Chapter 1, Introduction, this chapter, provides an overview of the purpose and functions of the DigitalPersona One Touch for CE SDK. It also specifies the target audience; describes the typographical, notational, and naming conventions used in the guide; identifies additional resources that may be helpful to you; lists the minimum system requirements and provides a compatibility matrix of supported DigitalPersona products.

Chapter 2, Installation & Deployment, contains instructions for installing the SDK and the RTE (runtime engine) that you will distribute with your application.

Chapter 3, Fingerprint Device Driver Interfaces, provides information on the functions used to manage the fingerprint reader.

Chapter 4, Fingerprint Recognition Engine Interface, defines the functions used for performing feature extraction.

Chapter 5, Matching Functions, defines the functions used to compare the registration template (obtained by processing pre-registration features) and verification features (extracted from the fingerprint scan), and calculate a score that indicates how likely it is that they come from the same finger.

Index, provides a way to quickly find information in the guide.

Document Conventions

This section defines the notational, typographical, and naming conventions used in this guide.

Notational Conventions

The following notational conventions are used to call attention to information of special importance:

NOTE: Notes provide reminders, tips, or suggestions that supplement the material included in this guide.

IMPORTANT: Important notations advise you of atypical system behavior or emphasize certain functionality.

WARNING: Warnings alert you to problems or side effects that can occur in specific situations.

Typographical Conventions

The following typographical conventions are used in this guide:

Typeface	Purpose	Example
Courier Bold	Used to indicate code that is input by the user or messages that are output by the system	Input: DPLMGetInfo() ; Output: No more entries are available.
<i>Italics</i>	Used for emphasis or to introduce new terms For online viewers, used to indicate potential links to URLs and other parts of the guide	DPIDEndSearch <i>must</i> be called on the same thread as the corresponding DPIDBeginXxx . <i>Duration</i> is the period of time for which a template certificate, once issued, is valid. Call DPIDCreateIdentificationSet before calling this function (<i>page 20</i>). For developers who are viewing this document online, text in italics may also indicate hyperlinks to other areas in this guide.
Bold	Used in running text for keystrokes and window and dialog box elements	Press Enter . Click the Info tab.

Naming Conventions

The DigitalPersona One Touch for CE SDK is also referred to as simply the SDK.

Additional Resources

You can refer to the resources in this section to assist you in using the One Touch for CE SDK.

Related Documentation

For information about	See
Fingerprint recognition, including the history and basics of fingerprint identification and the advantages of the DigitalPersona Fingerprint Recognition Engine	The DigitalPersona White Paper: Guide to Fingerprint Recognition (Fingerprint Guide.pdf) located in the Docs folder of the product package)
Late-breaking news about the product	The Readme.txt files provided in the root directory of the product package as well as in some subdirectories

Online Resources

To locate the	Go to
DigitalPersona Developer Connection Forum for DigitalPersona Developers	http://www.digitalpersona.com/webforums/
Latest updates for DigitalPersona software products	http://www.digitalpersona.com/support/downloads/software.php

System Requirements

This section lists the minimum hardware and software requirements needed to run the SDK on the development platform, and minimum requirements for the target platform.

Development Platform

- Pentium processor (or equivalent)
- Windows XP Professional or above

Target (CE) Platform

- X86-based processor with Microsoft® Windows® CE.NET 6.0
- USB port on the device that the fingerprint reader will be connected to
- 16 MB available memory

DigitalPersona Application Compatibility

DigitalPersona One Touch for CE SDK Version 1.3.0 fingerprint templates are interoperable with those generated by the following DigitalPersona products:

- DigitalPersona One Touch for Windows SDK/Fingerprint Recognition Software
- DigitalPersona Pro for AD (Workstation/Server/Kiosk)
- DigitalPersona Platinum SDK/Fingerprint Recognition Software

This chapter contains instructions for installing the various components of the One Touch for CE SDK.

Installing the SDK

To install the SDK on your development platform, copy the SDK folder from the product package to your hard drive. This folder includes the software libraries and sample applications.

This documentation is not included in the installation. You can copy it to your hard drive separately if you wish to.

Installing the Runtime

The SDK also includes a runtime which must be installed on the target machine for testing, and must be distributed with your application to enable fingerprint recognition functionality.

To install the DigitalPersona One Touch for CE runtime on the target hardware platform

1. Open the Runtime Install folder in the product package.
2. Open the folder corresponding to your target processor.
3. Connect the ActiveSync between the host PC and the target hardware and run Setup.exe from the product package.

Or, you can copy the .cab file from the product package and run it on the target hardware.

4. Follow the onscreen instructions to complete setup.
5. Plug a U.are.U 4000B Fingerprint Reader in the USB port of the target.

Overview

The DigitalPersona Windows CE.NET Fingerprint device driver (or simply driver in this document) provides two interface DLLs to the client application.

- IDeviceManager interface - for device control and management
- IDevice interface - for control and receiving of event and fingerprint data from the device.

The interfaces are provided in C++ class style. The first section describes the device manager interface.

IDeviceManager interface

The purpose of this interface is to provide an abstraction layer to the application for managing the fingerprint device's life cycle, receiving asynchronous events, and controlling the device. The interface component includes the following files:

- dpFDdevice.h
- DevTypes.h
- Platform.h
- dpDrvApi.lib
- dpDrvApi.dll

It exports all the necessary supporting data types, two C++ interface classes (or abstract classes) – IDevice (device interface) and IDeviceManager (device manager interface), and a C style function for obtaining the IDeviceManager interface pointer. The data type and interfaces classes are declared within the C++ name space DigitalPersona::Driver.

The client application first obtains the device manager interface IDeviceManager pointer, providing a callback to the driver for device plug and unplug notification. For each new device, the client application opens the device through the IDeviceManager interface method and receives a device interface IDevice pointer that represents that device. Then the application can receive an asynchronous event and controls the device through the device interface.

The client obtains this interface pointer by calling a C style function. The prototype of this C function is:

```
DPFD_DM_API DigitalPersona::Driver::IDeviceManager* C_CALL  
GetDeviceManagerInterface();
```

DPFD_DM_API is defined as __declspec(dllimport), and

C_CALL is defined as __cdecl,

DigitalPersona::Driver is namespace.

IDeviceManager methods

OpenDeviceManager

This method initiates the connection between the client application and the driver.

Syntax

```
virtual int OpenDeviceManager (
    IN FD_PNP_CALLBACK    PnpCB,    // Plug & Play CallBack
    IN void*    pvCtx    // Device Manager context
) = 0;
```

Parameters:

PnpCB	[in] Pointer to the application-defined callback function of type FD_PNP_CALLBACK. The callback is executed by the driver when a device is plugged or unplugged.
pvCtx	[in] Pointer to application data that will be passed back to the callback function.

Remarks

This method installs client's callback function PnpCB to handle the device plug and unplug (a.k.a. Plug&Play). The device manager can be opened only once system-wide. For example, if two applications try to open the device manager independently, only one will succeed. This method must be called before any other method.

Returns

eErrSuccess indicates success, others indicate error.

GetVersion

Retrieves the device manager software version, which contains major, minor, revision and build number information.

Syntax

```
virtual FD_VERSION GetVersion() = 0;
```

CloseDeviceManager

This method closes the device manager. The client should also call the Release() method for the final cleanup.

Syntax

```
virtual void CloseDeviceManager() = 0;
```

EnumerateDevices

This method allows the application to query the number of the devices found by the system and driver.

Syntax

```
virtual int EnumerateDevices (  
    OUT    FD_UID*    pDeviceUid,  
    IN unsigned int    uMaxNbOfUid,  
    OUT unsigned int*  puNbOfDevices  
) = 0;
```

Parameters:

PDeviceUid	[out] Application provided array of type FD_UID for storing device UIDs.
UMaxNbOfUid	[in] Maximum number of elements in the pDeviceUid array.
PuNbOfDevices	[out] Pointer to a variable that receives actual number of devices found.

Remarks

This method returns all the device IDs that have been added to the driver's internal list. It does not necessarily provide the UIDs of all the devices connected to the system at the time the call is made. The driver detects the device's existence via system calls.

Upon finding a device, the driver will initialize the device and add it to the list only when the device is initialized successfully. If this method is called while the device is being initialized, the count returned may not reflect the actual number of devices in the system. An alternative (and recommended) way to enumerate devices is to use the PnP callback function presented in the OpenDeviceManager() method.

Returns

eErrSuccess indicates success, others indicate error.

GetDeviceInfo

Retrieves device information by device's UID.

Syntax

```
virtual int GetDeviceInfo (  
    IN  FD_UID DeviceUid,  
    OUT FD_DEVICE_INFO& DeviceInfo  
) = 0;
```

Parameters:

DeviceUid	[in] Identifies the device whose information is to be retrieved.
DeviceInfo:	[out] Variable to store the information.

Returns

eErrSuccess indicates success, others indicate error.

OpenDevice

Opens a device and gets the device interface pointer.

Syntax

```
virtual int OpenDevice (  
    IN  FD_UID DeviceUid,  
    IN  FD_DEVICE_EVENT_CALLBACK EventCB,  
    IN void* pvCtx,  
    OUT IDevice** ppDevice  
) = 0;
```

Parameters:

DeviceUid	[in] Identifies the device to be opened.
EventCB	[in] Pointer to application-defined callback function of type FD_DEVICE_EVENT_CALLBACK. The callback is executed by the driver when the driver needs to send an event to the application.
pvCtx	[in] Pointer to application data (client context) that will be passed to the EventCB function.
ppDevice	[out] Pointer to the device interface pointer.

Remarks

After the device is opened, the application can use the device interface pointer to control or retrieve information from the device. The events, possibly including data, are delivered to the application through the EventCB callback function.

Returns

eErrSuccess indicates success, others indicate error.

CloseDevice

Close the device and no more events will be sent to the application from this device. The driver will release the memory allocated for pDevice.

Syntax

```
virtual int CloseDevice(IN IDevice* pDevice) = 0;
```

Release

Release the device manager interface pointer. This releases the allocated memory.

Syntax

```
virtual void Release() = 0;
```

IDevice interface

While the device manager interface discussed in the previous pages handles the creation and destruction of device objects, the device interface is designed for controlling and receiving event and fingerprint data from an individual device. A device interface pointer represents a unique fingerprint device connected to the system and an application obtains it through the device manager interface method `OpenDevice()`.

IDevice methods

GetParameter

This method is used to read internal device data. Presently it is not for public use.

Syntax

```
virtual int GetParameter ( IN OUT    FD_PPARAMETER  pParam ) = 0;
```

SetParameter

This method is used to set internal device data. Presently it is not for public use.

Syntax

```
virtual int ( IN OUT    FD_PPARAMETER  pParam ) = 0;
```


Overview

The dpFtrEx module contains code to perform feature extraction, i.e. the process of deriving feature data from the fingerprint scan in a form that is suitable for comparing prints and determining if they come from the same finger.

Header file: dpFtrEx.h, Language: ANSI C, Notes: prefix FX_

Module Initialization, Termination and Settings

FX_getVersionInfo

Returns the software version of the feature extraction module.

Syntax

```
Void FX_getVersionInfo ( FT_VERSION_INFO_PT versionInfoPt )
```

Parameter

versionInfoPt	[out] Pointer to the version information for the dpFtrEx module.
---------------	--

Returns

None.

FX_init

Initializes the feature extraction module, and must be called before any other functions are called.

Syntax

```
FT_RETCODE fx_init (void)
```

Returns

<i>FT_OK</i>	Initialization successful.
<i>FT_ERR_BAD_INI_SETTING</i>	An error occurred opening, getting info from, or reading a value from a registry key.
<i>FT_ERR_NO_MEMORY</i>	Memory was not allocated due to failure of the malloc or calloc function.

FX_createContext

Creates a feature extraction context.

Syntax

```
FT_RETCODE FX_createContext ( FT_HANDLE *fxContext )
```

Returns

<i>FT_OK</i>	Initialization successful.
<i>FT_ERR_NO_INIT</i>	Feature Extraction module not initialized.
<i>FT_ERR_NO_MEMORY</i>	Memory was not allocated due to failure of the malloc or calloc function.

FX_closeContext

Closes a feature extraction context.

Syntax

```
FT_RETCODE FX_closeContext ( FT_HANDLE fxContext )
```

Returns

<i>FT_OK</i>	Initialization successful.
<i>FT_ERR_NO_INIT</i>	Feature Extraction module not initialized.
<i>FT_ERR_INVALID_CONTEXT</i>	The specified context is not valid.

FX_terminate

Terminates the feature extraction module, and releases all resources.

Syntax

```
FT_RETCODE fx_terminate (void)
```

Returns

<i>FT_OK</i>	Closing of context succeeded.
<i>FT_WRN_NO_INIT</i>	Warning. Feature extraction module has not been initialized.

Feature Extraction

The feature extraction module maintains one or more contexts for each caller thread. A context can be created by calling `FX_createContext`, and eventually released with `FX_closeContext`. A handle to the context has to be passed to `FX_extractFeatures` and `FX_getDisplayImage`.

FX_getFeaturesLen

Returns the minimum and recommended length of the features.

Syntax

```
FT_RETCODE FX_getFeaturesLen (
    FT_FTR_TYPE ftrType,
    int *recommendedFtrLen,
    int *minFtrLen,
)
```

Parameters:

ftrType	[in] Specifies whether to get the length for verification or registration features. Valid values are: <ul style="list-style-type: none"> ■ FT_PRE_REG_FTR - Registration ■ FT_VER_FTR - Verification.
recommendedFtrLen	[out] The recommended length of the features for best performance, or NULL
minFtrLen	[out] The minimum length of the features or NULL.

Returns

FT_OK	Initialization successful.
FT_ERR_NO_INIT	Warning. Feature extraction module has not been initialized.
FT_ERR_INVALID_PARAM	Invalid features type.

FX_extractFeatures

Extracts the features from the given scan (image), which is passed as one of the arguments.

Syntax

```
FT_RETCODE FX_extractFeatures (
    FT_HANDLE fxContext,
    int devImgSize,
    FT_IMAGE_PT devImgPt,
    FT_FTR_TYPE featuresType,
    int featuresLen,
    FT_BYTE *features,
    FT_IMG_QUALITY_PT imgQualityPt,
    FT_FTR_QUALITY_PT ftrQualityPt,
    FT_BOOL *extractOK
)
```

Parameters:

fxContext	[in] Handle of the context to use. This image reprocessing handle allows the function to communicate with the preprocessing module to get a device independent image. The context handle is obtained from FX_createContext.
devImgSize	[in] Device image size (in bytes).
devImgPt	[in] Device image.
featuresType	[in] Specifies whether the features extracted are going to be used for verification or registration. Valid values are: <ul style="list-style-type: none"> ■ FT_PRE_REG_FTR - Registration ■ FT_VER_FTR - Verification
featuresLen	[in] Specifies the length (in bytes) of the features buffer. Use FX_getFeaturesLen to get information about the features length to use.
features	[out] Extracted features
imgQualityPt	[out] Pointer to a value representing the quality of the specified image. If the image quality is not equal to FT_GOODIMG, the extraction is not performed and ftrQualityPt is set to FT_UNKNOWNFTRQUALITY.
ftrQualityPt	[out] Pointer to a value representing the quality of the specified features.
extractOK	[out] If the extraction is successful, extractOK is set to FT_TRUE.

Returns

<i>FT_OK</i>	Initialization successful.
<i>FT_ERR_NO_INIT</i>	Warning. Feature extraction module has not been initialized.
<i>FT_ERR_NO_MEMORY</i>	Memory was not allocated due to failure of the malloc or calloc function.
<i>FT_ERR_INVALID_CONTEXT</i>	The specified context is not valid.
<i>FT_ERR_INVALID_PARAM</i>	Invalid features type.
<i>FT_ERR_UNKNOWN_DEVICE</i>	The device us unknown.

FX_getDisplayImage

Returns an image prepared for display. It may involve resizing (to the requested size), changing the number of intensity levels, rotation and other image processing needed for the image to look good during display.

The input is an image that comes from device. It is the same as that of FX_extractFeatures function. The output is a grey scale, 8 bits per pixel buffer.

Syntax

```
FT_RETCODE FX_getDisplayImage (
    FT_HANDLE fxContext,
    const FT_IMAGE_PT devImgPt,
    const FT_IMAGE_SIZE_PT pImageSize,
    const FT_BOOL bRotation,
    const int numIntensityLevels,
    FT_IMAGE_PT *pImageBuffer,
)
```

Parameters:

fxContext	[in] Handle of the context to use. The image preprocessing handle allows the function to communicate with the preprocessing module to get a device independent image. The context handle is obtained from FX_createContext.
devImgPt	[in] Device image.
devImgSize	[in] Device image size (in bytes).
bRotation	[in] Rotation flag.
numIntensityLevels	[in] Requested number of intensity levels.
pImageBuffer	[in out] Output image (grey scale, 8 bits per pixel)

Returns

<i>FT_OK</i>	Initialization successful.
<i>FT_ERR_NO_INIT</i>	Warning. Feature extraction module has not been initialized.
<i>FT_ERR_NO_MEMORY</i>	Memory was not allocated due to failure of the malloc or calloc function.
<i>FT_ERR_INVALID_CONTEXT</i>	The specified context is not valid.
<i>FT_ERR_INVALID_PARAM</i>	Invalid features type.
<i>FT_ERR_UNKNOWN_DEVICE</i>	The device us unknown.

Overview

The dpMatch module contains code that compares the registration template (obtained by processing pre-registration features) and verification features (extracted from the fingerprint scan) and calculates a score that indicates how likely it is that they come from the same finger. Most of the functions must be called in a particular context, which is specified by passing a context handle as the first argument.

Header file: dpMatch.h

Language: ANSI C Notes: prefix MC_

Module Initialization, Termination and Settings

MC_init

Initializes the matching module, and must be called before any other functions are called.

Syntax

```
FT_RETCODE MC_init (void)
```

Returns

<i>FT_OK</i>	Initialization successful.
<i>FT_ERR_BAD_INI_SETTING</i>	An error occurred opening, getting info from, or reading a value from a registry key.
<i>FT_ERR_NO_MEMORY</i>	Memory was not allocated due to failure of the malloc or calloc function.

MC_terminate

Closes the dpMatch module and releases all resources.

Syntax

```
FT_RETCODE MC_terminate(void)
```

Returns

FT_OK If function is successful.

MC_createContext

Creates a context, and returns a handle for it.

Syntax

```
FT_RETCODE MC_createContext ( FT_HANDLE *mcContext )
```

Parameter

mcContext	[out] Context handle if context creation is successful.
-----------	---

Returns

FT_OK If function is successful.

MC_closeContext

Closes the context created by MC_createContext and frees the allocated resources.

Syntax

```
FT_RETCODE MC_closeContext ( FT_HANDLE mcContext )
```

Parameter

mcContext	[in] Handle of the context to close. The context handle is obtained from MC_createContext.
-----------	--

Returns

FT_OK If function is successful.

MC_getVersionInfo

Returns the software version of the feature extraction module.

Syntax

```
void MC_getVersionInfo ( FT_VERSION_INFO_PT versionInfoPt )
```

Parameter

versionInfoPt	[out] Returns a pointer to the version information of the dpMatch module.
---------------	---

Returns

FT_OK If function is successful.

MC_getSettings

Gets the current dpMatch module settings.

Syntax

```
FT_RETCODE MC_getSettings (MC_SETTINGS_PT settingsPt )
```

Parameter

settingsPt	[out] Returns a pointer to the settings information.
------------	--

Remarks

The dpMatch module has settings, which are returned by MC_getSettings in a structure of type MC_SETTINGS. It provides the number of pre-registration features required for registration. The settings are read only.

Returns

FT_OK If function is successful.

MC_setSecurityLevel

Sets the security level of the matching module.

Syntax

```
FT_RETCODE MC_setSecurityLevel (
    FT_HANDLE mcContext,
    FT_FA_RATE fa_rate,
)
```

Parameters:

mcContext	[in] The context handle obtained from MC_createContext.
fa_rate	[in] The False Accept Rate to set for the context. The lower fa_rate is, the higher the security will be, but also the higher the false reject rate will be. Suggested values of fa_rate are HIGH_SEC_FA_RATE, MED_SEC_FA_RATE (the default) and LOW_SEC_FA_RATE.

Remarks

MC_setSecurityLevel can be used to modify the security level, according to the accuracy required by the application.

Returns

FT_OK	Function completed successfully.
FT_ERR_NO_INIT	Matching module has not been initialized.
FT_ERR_INVALID_CONTEXT	The specified context is not valid.
FT_ERR_INVALID_PARAM	The fa_rate \leq 0.0 or the fa_rate \geq 100.0, or the named context is invalid.
FT_WRN_INTERNAL	The desired fa_rate was unacceptably high, and was reduced to the highest acceptable value.

MC_getSecurityLevel

Gets the security level of the given context in terms of the False Accept Rate (FAR).

Syntax

```
FT_RETCODE MC_getSecurityLevel (
    FT_HANDLE mcContext,
    FT_FA_RATE *fa_rate,
)
```

Parameters:

mcContext	[in] The context handle obtained from MC_createContext.
fa_rate	[out] Percentage of false accepts (FAR).

Remarks

Both registration and verification are performed with a given accuracy. MC_getSecurityLevel returns the current security level.

Feature registration and verification

MC_getFeaturesLen

Returns the minimum and recommended length of the features for a give feature type.

Syntax

```
FT_RETCODE MC_getFeaturesLen (
    FT_FTR_TYPE ftrType,
    FT_REG_OPTIONS mcRegOptions,
    int *recommendedFtrLen,
    int *minFtrLen,
)
```

Parameters:

ftrType	[in] Specifies whether to get the length for verification or registration features. Valid values are: FT_PRE_REG_FTR - Registration FT_VER_FTR - Verification.
mcRegOptions	[in] Registration options. Can be 0 or a bitwise mask of different options (See Remarks section below.).
*recommendedFtrLen	[out] Or NULL.
*minFtrLen	[out] Or NULL.

Remarks

Different options for registration may be set with mcRegOptions by combining bit masks with the bitwise OR operator.

Currently there are four options (bit masks) available:

- FT_DEFAULT_REG: enables the default combination of options which ensures the optimal performance. This default set of options may be changed in future releases. Currently XTF registration is set by default.
- FT_DISABLE_DEFAULT_REG: disables the implicit use of the default registration options. It gives the caller complete control over the registration options used. It can be used in conjunction with the other bit masks. In this case, the options explicitly set in the resulting mask will be used for registration. For example, using the FT_DISABLE_DEFAULT_REG | FT_ALLOW_LEARNING bit mask will guarantee that only learning will be set - regardless of the default options implicitly set by current engine.
- FT_ENABLE_XTF_REG: enables the XFT registration.
- FT_ALLOW_LEARNING: enables learning during fingerprint verification.

Returns

FT_OK If function is successful.

MC_verifyFeaturesEx

Performs verification of registration and verification features.

Syntax

```
FT_RETCODE MC_verifyFeaturesEx (
    FT_HANDLE mcContext,
    int regFeaturesLen,
    [in out] *regFeatures,
    int verFeaturesLen,
    FT_BYTE *verFeatures,
    FT_BOOL doLearning,
    FT_BOOL *regFeaturesChanged,
    FT_BYTE reserved[FT_KEY_LEN],
    FT_VER_SCORE_PT scorePt,
    doubleFT_VER_SCORE_PT *falseAcceptProbability,
    FT_BOOL *verified,
)
```

Parameters

mcContext	[in] The context handle obtained from MC_createContext.
regFeaturesLen	[in] Specifies how much space to allocate for the storage of registration features in regFeatures.
regFeatures	[in out] Fingerprint template data will be used for verification.
verFeaturesLen	[in] Fingerprint feature set size.
verFeatures	[in] Fingerprint feature set needing to be verified.
doLearning	<p>[in] If doLearning is set to TRUE, this function is allowed to change the registration features, which would set regFeaturesChanged to TRUE.</p> <p>The registration features might contain learning data, in which case that data will be used, but it will not be modified unless doLearning is true.</p> <p>Also, if TRUE, an error will be returned if the registration features don't contain learning data (which is determined by the value of mcRegOptions passed to MC_generateRegFeatures at the time the features were generated).</p>

Parameters (continued)

regFeaturesChanged	[out] TRUE if registration features have been changed as a result of learning, FALSE or NULL if not changed.
reserved[FT_KEY_LEN]	[out] Obsolete. Should be NULL.
scorePt	[out] Contains pointer to a value that indicates how well the prints matched if they did match. If they did not match, the value is NULL.
falseAcceptProbability	[out] Returns a value indicating the probability that non-matching prints would be reported as verified.
verified	[out] TRUE if registration features and verification features matched. FALSE if they did not match.

Remarks

MC_verifyFeaturesEx compares a registration template to verification features and returns TRUE if the matching score is high enough (given the current security level). If learning is enabled, it attempts to “learn” and may update the registration template. The registration and verification features must be passed, along with their lengths.

Returns

FT_OK	Function completed successfully.
FT_ERR_NO_INIT	Matching module not initialized.
FT_ERR_NO_MEMORY	The calloc function failed.
FT_WRN_LEARNING_IMPOSSIBLE	Learning was attempted but was impossible to perform.
FT_ERR_BAD_INI_SETTING	There was a problem opening a registry key or reading a value from it.
FT_ERR_INVALID_BUFFER	The output buffer is NULL or of insufficient length.
FT_ERR_VERS_MISMATCH	Version mismatch.
FT_ERR_INVALID_PARAM	Invalid data for given mode, or MC_NUREGFP != numCandidates.

MC_generateRegFeatures

Call at registration time to create the registration features for a finger.

Syntax

```
FT_RETCODE MC_generateRegFeatures (
    FT_HANDLE mcContext,
    FT_REG_OPTIONS mcRegOptions,
    int numPreRegFeatures,
    int preRegFeaturesLen,
    FT_BYTE *preRegFeatures[],
    int regFeaturesLen,
    FT_BYTE *regFeatures,
    FT_BYTE reserved[FT_KEY_LEN],
    FT_BOOL regSucceeded,
)
```

Parameters:

mcContext	[in] The context handle obtained from MC_createContext.
mcRegOptions	[in] Specifies registration options. Learning: To include learning data in the features, set FT_ALLOW_LEARNING through bitwise OR operator. If learning is to be done at any time in the future, this option must be set. Can also be 0.
numPreRegFeatures	[in] Number of input (i.e. pre-registration) fingerprints. MC_NUMREGFP must be passed for numPreRegFeatures. *** numPreRegFeatures (field of MC_SETTINGS structure obtained with MC_getSettings function) must be passed for numPreRegFeatures. ***
preRegFeaturesLen	[in] Specifies the length of the buffer needed to store the features of a single pre-registration print.
preRegFeatures[]	[in] preRegFeatures must contain an array of MC_NUMREGFP pointers to pre-registration features for the finger being registered.
regFeaturesLen	[in] Specifies how much space to allocate for the storage of registration features in regFeatures.
regFeatures	[out] Features produced if registration succeeded.
reserved[FT_KEY_LEN]	[out] Obsolete. Should be NULL.
regSucceeded	[out] TRUE if registration was successful.

Remarks

MC_generateRegFeatures requires as input the features extracted from multiple samples of a single finger. (These are known as pre-registration features, and are obtained with the function FT_extractFeatures.)

This function calculates matching scores between all the pairs of samples, and if they are determined to be good enough, it constructs and returns the registration features that will be used to verify the user in the future.

If the registration is successful, then the new registration features are written to regFeatures, and *regSucceeded is true. If the registration fails because the input features don't match well enough, *regSucceeded is false.

A registration template is created by MC_generateRegFeatures. It requires the pre-registration features extracted from multiple samples of a single finger as input. It constructs and returns the registration template that will be used to verify the fingerprint in the future.

The XTF (eXtended Template Format) registration templates contain additional information needed for a more accurate verification. The template size may be up to four times bigger than that of the basic template (about 1200 instead of 300 bytes). The matching time may slightly increase. If you do not have storage limitations, it is strongly recommended to use XTF registration.

During verification, the recognition engine may “learn” additional information about the fingerprint, and may update the registration template, so that future verifications will be performed with higher accuracy. In order to use this feature, learning must be enabled during registration.

Different options for registration may be set with mcRegOptions by combining bit masks with the bitwise OR operator. Currently there are four options (bit masks) available:

1. FT_DEFAULT_REG: enables the default combination of options which ensures the optimal performance. This default set of options may be changed in future releases. Currently XTF registration is set by default.
2. FT_DISABLE_DEFAULT_REG: disables the implicit use of the default registration options. It gives the caller complete control over the registration options used. It can be used in conjunction with the other bit masks. In this case, the options explicitly set in the resulting mask will be used for registration. For example, using the FT_DISABLE_DEFAULT_REG | FT_ALLOW_LEARNING bit mask will guarantee that only learning will be set - regardless of the default options implicitly set by current engine.
3. FT_ENABLE_XTF_REG: enables the XFT registration.
4. FT_ALLOW_LEARNING: enables learning during fingerprint verification.

Returns

FT_OK	Function completed successfully.
FT_ERR_NO_INIT	Matching module not initialized.
FT_ERR_NO_MEMORY	The calloc function failed.
FT_ERR_BAD_INI_SETTING	There was a problem opening registry key or reading a value from it.

FT_ERR_INVALID_BUFFER	The output buffer is NULL or of insufficient length.
FT_ERR_VERS_MISMATCH	Version mismatch.
FT_ERR_INVALID_PARAM	Invalid data for given mode, or MC_NUREGFP != numCandidates.
FT_ERR_INTERNAL	Data out of bounds, or incompatible data.

Index

A

- additional resources 4
 - online resources 4
 - related documentation 4
- Application Compatibility Matrix 5
- audience for this guide 2

B

- bold typeface, uses of 3

C

- chapters, overview of 2
- CloseDevice 11
- CloseDeviceManager 9
- conventions, document
 - See document conventions
- Courier typeface, uses of 3

D

- DevTypes.h 7
- DigitalPersona support website 4
- distributing your application 6
- document conventions 3
 - notational 3
 - typographical 3
- documentation, related 4
- dpDrvApi.dll 7
- dpDrvApi.lib 7
- dpFDdevice.h 7

E

- EnumerateDevices 9

F

- Feature registration and verification 23
- FX_closeContext 14
- FX_createContext 14
- FX_extractFeatures 16
- FX_getDisplayImage 17
- FX_getFeaturesLen 15
- FX_getVersionInfo 13
- FX_init 13
- FX_terminate 14

G

- GetDeviceInfo 10
- GetParameter 12
- GetVersion 8

I

- IDevice interface 12
- IDevice methods 12
- IDeviceManager interface 7
- IDeviceManager methods 8
- installing the runtime 6
- installing the SDK 6
- introduction to this guide 1
- italics typeface, uses of 3

M

- Matching Functions 19
- MC_closeContext 20
- MC_createContext 20
- MC_generateRegFeatures 26
- MC_getFeaturesLen 23
- MC_getSecurityLevel 22
- MC_getSettings 21
- MC_getVersionInfo 20
- MC_init 19
- MC_setSecurityLevel 21
- MC_terminate 19
- MC_verifyFeaturesEx 24

N

- name space 7
- notational conventions 3
- note notation, defined 3

O

- online resources
- OpenDevice 10
- OpenDeviceManager 8
- overview of chapters 2

P

- Platform.h 7

R

Release 11

requirements, system

See system requirements

resources, online

See online resources

S

SetParameter 12

system requirements

T

target audience for this guide 2

typefaces, uses of

bold 3

Courier 3

italics 3

typographical conventions 3

W

warning notation, defined 3