DigitalPersona, Inc.

One Touch® for Windows® SDK
C/C++ Edition

Version 1.4

Developer Guide

digital**Persona**.

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

Document Publication Date: May 22, 2009

Table of Contents

Introduction 1

The One Touch® for Windows SDK is a software development tool that enables developers to integrate fingerprint biometrics into a wide set of Microsoft® Windows®-based applications, services, and products. The tool enables developers to perform basic fingerprint biometric operations: capturing a fingerprint from a DigitalPersona fingerprint reader, extracting the distinctive features from the captured fingerprint sample, and storing the resulting data in a template for later comparison of a submitted fingerprint with an existing fingerprint template.

In addition, the One Touch for Windows SDK enables developers to use a variety of programming languages in a number of development environments to create their applications. The product includes detailed documentation and sample code that can be used to guide developers to quickly and efficiently produce fingerprint biometric additions to their products.

The One Touch for Windows SDK builds on a decade-long legacy of fingerprint biometric technology, being the most popular set of development tools with the largest set of enrolled users of any biometric product in the world. Because of its popularity, the DigitalPersona® Fingerprint Recognition Engine software — with its high level of accuracy — and award-winning U.are.U® Fingerprint Reader hardware have been used with the widest-age, hardest-to-fingerprint demographic of users in the world.

The One Touch for Windows SDK has been designed to authenticate users on the Microsoft® Windows Vista® and Microsoft® Windows® XP operating systems running on any of the x86-based platforms. The product is used with DigitalPersona fingerprint readers in a variety of useful configurations: standalone USB peripherals, modules that are built into customer platforms, and keyboards. The DigitalPersona One Touch I.D. SDK product can also be implemented along with the One Touch for Windows SDK product to add fast fingerprint identification capability to a developer's design.

Fingerprint Authentication on a Remote Computer

This SDK includes transparent support for fingerprint authentication through Windows Terminal Services (including Remote Desktop Connection) and through a Citrix connection to Metaframe Presentation Server using a client from the Citrix Presentation Server Client package.

Through Remote Desktop or a Citrix session, you can use a local fingerprint reader to log on to, and use other installed features of, a remote machine running your fingerprint-enabled application.

The following types of Citrix clients are supported:

- Program Neighborhood
- Program Neighborhood Agent
- Web Client

Note that to take advantage of this feature, your fingerprint-enabled application must run on the Terminal Services or Citrix server, not on the client. If you are developing a Citrix-aware application, see additional information in the *Developing Citrix-aware applications* chapter on page *85*.

Target Audience

This guide is for developers who have a working knowledge of the C or C++ programming language.

Chapter Overview

Chapter 1, Introduction, this chapter, describes the audience for which this guide is written; defines the typographical and notational conventions used throughout this guide; identifies a number of resources that may assist you in using the One Touch for Windows SDK: C/C++ Edition; identifies the minimum system requirements needed to run the One Touch for Windows SDK: C/C++ Edition; and lists the DigitalPersona products and fingerprint templates supported by the One Touch for Windows SDK: C/C++ Edition.

Chapter 2, *Quick Start*, provides a quick introduction to the One Touch for Windows SDK: C/C++ Edition using one of the sample applications provided as part of the SDK.

Chapter 3, *Installation*, contains instructions for installing the SDK and the RTE and identifies the files and folders that are installed on your hard disk.

Chapter 4, *Overview*, introduces One Touch for Windows SDK: C/C++ Edition terminology and concepts, shows how data flows among the various One Touch for Windows SDK: C/C++ Edition components, and includes workflow diagrams and explanations of the One Touch for Windows: C/C++ Edition API functions used to perform the operations in the workflows.

Chapter 5, *Core API Reference*, defines the functions, data structures, and type definitions that are part of the One Touch for Windows: C/C++ Edition Core API.

Chapter 6, *User Interface API Reference*, defines the functions and enumerations of the User Interface API, a high-level wrapper providing a premade user interface and access to the full functionality of the Core API through a small number of simple functions.

Chapter 7, *Events Notifications and Return Codes*, defines the codes returned by the One Touch for Windows: C/C++ Edition API functions.

Chapter 9, *Redistribution*, identifies the files that you may distribute according to the End User License Agreement (EULA) and lists the functionalities that you need to provide to your end users when you develop products based on the One Touch for Windows: C/C++ Edition API.

Appendix A, *Setting the False Accept Rate*, provides information about determining and using specific values for the FAR and evaluating and testing achieved values.

Appendix B, *Platinum SDK Enrollment Template Conversion*, contains sample code for converting Platinum SDK registration templates for use with the One Touch for Windows SDK: C/C++ Edition.

A glossary and an index are also included for your reference.

Document Conventions

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

Notational Conventions

The following notational conventions are used throughout this guide:

NOTE: Notes provide supplemental reminders, tips, or suggestions.

IMPORTANT: Important notations contain significant information about system behavior, including 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 computer programming code	The only valid value for this field is **`TRUE`**. Initialize the licensing library by calling the **`DPFPInit`** function.
Italics	Used for emphasis or to introduce new terms For developers who are viewing this document online, text in italics may also indicate hypertext links to other areas in this guide.	*Duration* is the period of time for which a template certificate, once issued, is valid. Call **`DPIDCreateIdentificationSet`** before calling this function (*page 20*).
Bold	Used for keystrokes and window and dialog box elements	Press **Enter**. Click the **Info** tab.

Additional Resources

You can refer to the resources in this section to assist you in using the One Touch for Windows SDK: C/C++ Edition.

Related Documentation

Subject	Document
Fingerprint recognition, including the history and basics of fingerprint identification and the advantages of DigitalPersona's Fingerprint Recognition Engine	The DigitalPersona White Paper: Guide to Fingerprint Recognition. The file, Fingerprint Guide.pdf, is located in the Docs folder in the software package, and is not automatically installed on your computer as part of the setup process.
Late-breaking news about the product	The Readme.txt files provided in the root directory in the SDK software package as well as in some subdirectories

Online Resources

Web site name	URL
DigitalPersona Developer Connection Forum for peer-to peer interaction between 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 software and hardware requirements needed to run the One Touch for Windows SDK: C/C++ Edition.

- x86-based processor or better
- Microsoft® Windows® XP, 32-bit and 64-bit versions; Microsoft® Windows® XP Embedded, 32-bit version[1]; or Microsoft® Windows Vista®, 32-bit and 64-bit versions
- USB connector on the computer where the fingerprint reader is to be connected
- DigitalPersona U.are.U 4000B or U.are.U 4500 fingerprint reader

Supported DigitalPersona hardware Products

The One Touch for Windows SDK: C/C++ Edition supports the following DigitalPersona hardware products:

- DigitalPersona U.are.U 4000B/4500 or later fingerprint readers and modules
- DigitalPersona U.are.U Fingerprint Keyboard

---

1. A list of DLL dependencies for installation of your application on Microsoft Windows XP Embedded, One Touch for Windows XPE Dependencies.xls, is located in the Docs folder in the SDK software package.

Fingerprint Template Compatibility

Fingerprint templates produced by all editions of the One Touch for Windows SDK are also compatible with the following DigitalPersona SDKs:

- Gold SDK

- Gold CE SDK

- One Touch for Linux SDK, all distributions

NOTE: Platinum SDK enrollment templates must be converted to a compatible format to work with these SDKs. See Appendix B on *page 93* for sample code that converts Platinum SDK templates to this format.

Quick Start 2

This chapter provides a quick introduction to the One Touch for Windows SDK: C/C++ Edition using one of the sample applications provided as part of the One Touch for Windows SDK.

The application is a sample Microsoft® Visual C++® project that demonstrates the functionality of the user interfaces supported by the One Touch for Windows SDK: C/C++ Edition User Interface API.

Quick Concepts

The following definitions will assist you in understanding the purpose and functionality of the sample application that is described in this section.

Enrollment—The process of capturing a person's fingerprint four times, extracting the features from the fingerprints, creating a fingerprint template, and storing the template for later comparison.

Verification—The process of comparing a captured fingerprint to a fingerprint template to determine whether the two match.

Unenrollment—The process of deleting a fingerprint template associated with a previously enrolled fingerprint.

For further descriptions of these processes, see Chapter 4 on *page 17*.

Install the Software

To install the One Touch for Windows SDK: C/C++ Edition

1. In the SDK folder in the SDK software package, open the Setup.exe file, and then click **Next**.

2. Follow the installation instructions as they appear.

3. Restart your computer.

Connect the Fingerprint Reader

Conenct the fingerprint reader to the USB connector on the system where you installed the SDK.

Using the Sample Application

By performing the exercises in this section, you will

- Start the sample application
- Enroll a fingerprint
- Verify a fingerprint
- Unenroll (delete) a fingerprint
- Exit the sample application

To start the sample application
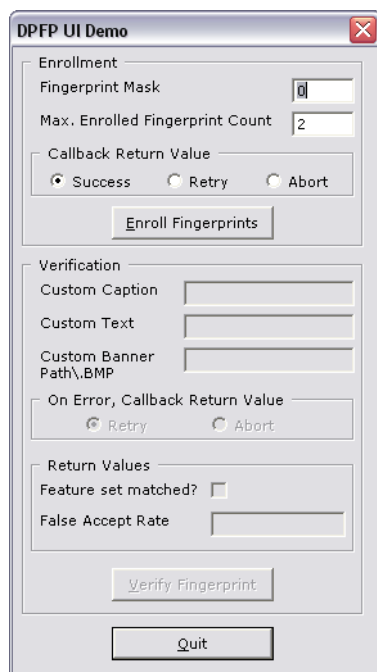
1. Open the UIVBDemo.exe file.

   It is located in the *<destination folder>*One Touch SDK\C++ Samples\ DPFP UI Demo\Release folder.

2. The **DPFPUI Demo** dialog box appears.

Enrolling a fingerprint consists of scanning your fingerprint four times using the fingerprint reader.

To enroll a fingerprint

1. In the **DPFPUI Demo** dialog box, click **Enroll Fingerprints**.

    The **Fingerprint Enrollment** dialog box appears.
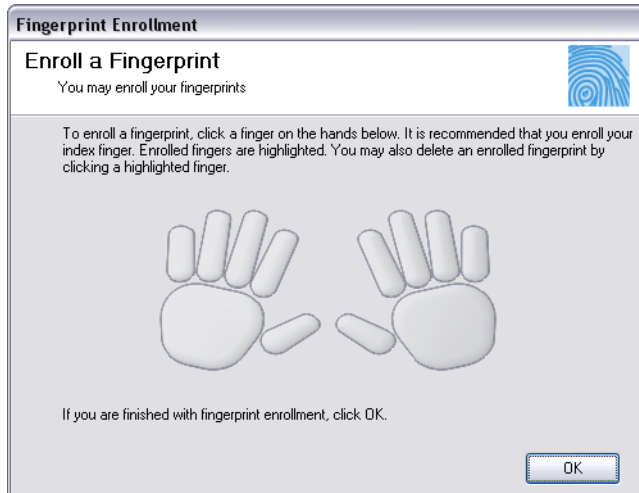
2. On the right "hand," click the index finger.

    A second **Fingerprint Enrollment** dialog box appears.
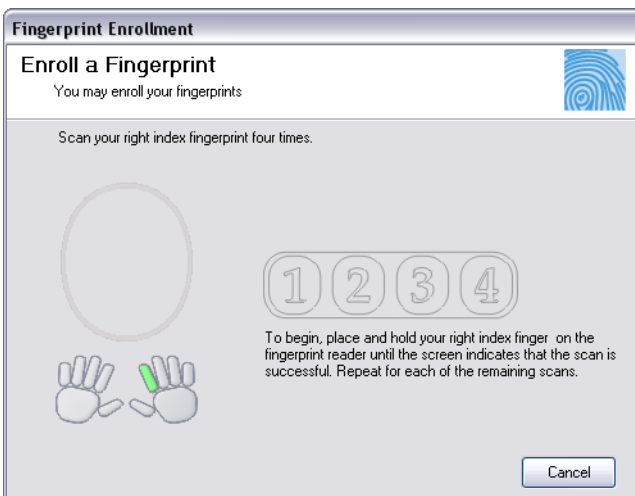
3. Using the fingerprint reader, scan your right index fingerprint.

4. Repeat step 3 until the **Success** message appears.

5. In the message box, click **OK**.

   The **Enrollment was successful** message appears.

6. Click **OK**.

To verify a fingerprint

1. In the **DPFPUI Demo** dialog box, click **Verify Fingerprint**.

   The **Verify Your Identify** dialog box appears.

2. Using the fingerprint reader, scan your right index fingerprint.

   The **Success** message appears, which indicates that your fingerprint was verified.
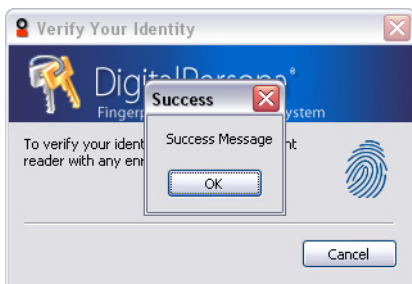
3. In the message box, click **OK**.

4. Using the fingerprint reader, scan your right middle fingerprint.

   The **Retry** message appears, which indicates that your fingerprint was not verified.
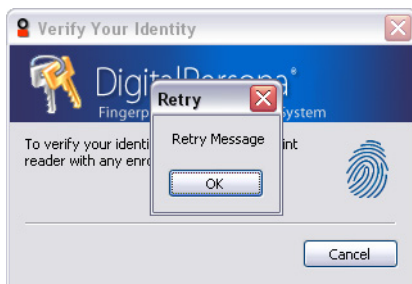
5. In the message box, click **OK**.

6. Click **Cancel**.

To unenroll (delete) a fingerprint

1. In the **DPFPUI Demo** dialog box, click **Enroll Fingerprints**.

   The **Fingerprint Enrollment** dialog box appears, indicating that you have enrolled your right index fingerprint.
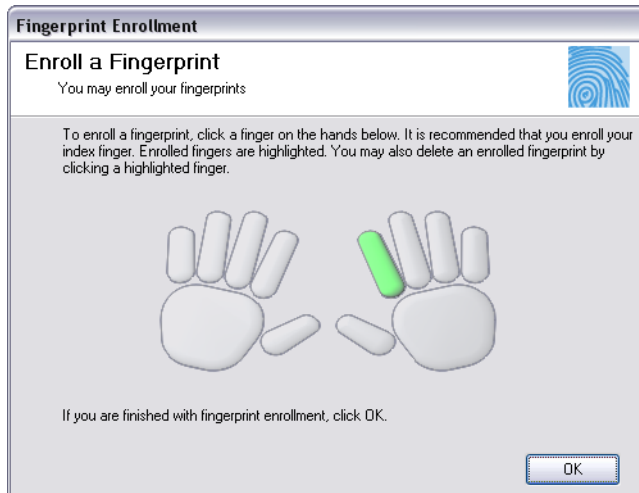
2. In the right "hand," click the green index finger.

   A message box appears, asking you to verify the deletion.

3. In the message box, click **Yes**.

    The **Success** message appears.

4. In the message box, click **OK**.

    The **Fingerprint Deleted** message appears.

    The right index finger is no longer green, indicating that the fingerprint associated with that finger is not enrolled (has been deleted).

To exit the application

■ In the **DPFPUI Demo** dialog box, click **Quit**.

Installation 3

This chapter contains instructions for installing the various components of the One Touch for Windows SDK: C/C++ Edition and identifies the files and folders that are installed on your hard disk.

The following two installations are located in the SDK software package:

- SDK, which you use in developing your application. This installation is located in the SDK folder.
- RTE (runtime environment), which you must provide to your end users to implement the One Touch for Windows SDK: C/C++ Edition interfaces, objects, methods, and properties. This installation is located in the RTE folder. (The RTE installation is also included in the SDK installation.)

Installing the SDK

To install the One Touch for Windows SDK: C/C++ Edition

1. In the SDK folder in the SDK software package, open the Setup.exe file, and then click **Next**.

2. Follow the installation instructions as they appear.

3. Restart your computer.

Table 1 describes the files and folders that are installed in the *<destination folder>* folder on your hard disk. The RTE files and folders, which are listed in Table 2 on *page 15*, are also installed on your hard disk.

NOTE: All installations share the DLLs and the DPHostW.exe file that are installed with the C/C++ edition. Additional product-specific files are provided for other editions.

Table 1. One Touch for Windows SDK: C/C++ Edition installed files and folders

Folder	File	Description
One Touch SDK\C-C++\Docs	One Touch for Windows SDK C-C++ Developer Guide.pdf	DigitalPersona One Touch for Windows SDK: C/C++ Edition Developer Guide
One Touch SDK\C-C++\ Include	dpDefs.h DPDevClt.h dpFtrEx.h dpMatch.h dpRCodes.h dpUIApi.h	Header files used by all of the One Touch for Windows SDK APIs

Table 1. One Touch for Windows SDK: C/C++ Edition installed files and folders *(continued)*

Folder	File	Description
One Touch SDK\C-C++\Lib	DPFPApi.lib DPFpUI.lib dpHFtrEx.lib dpHMatch.lib	Import libraries used by the One Touch for Windows SDK: C/C++ Edition API
One Touch SDK\C-C++\C++ Samples\DPFP UI Demo	This folder contains a sample Microsoft® Visual C++® project that demonstrates the functionality of the user interfaces supported by the One Touch for Windows SDK: C/C++ Edition User Interface API.	
One Touch SDK\C-C++\C++ Samples\Enrollment Sample Code	This folder contains a sample Microsoft Visual C++ project that shows how to use the One Touch for Windows: C/C++ Edition Core API for performing fingerprint enrollment and fingerprint verification.	

Installing the Runtime Environment (RTE)

When you develop a product based on the One Touch for Windows SDK: C/C++ Edition, you need to provide the redistributables to your end users. These files are designed and licensed for use with your application. You may include the installation files located in the RTE\Install folder in your application or you may incorporate the redistributables directly into your installer. You may also use the merge modules located in the Redist folder in the SDK software package to create your own MSI installer. (See *Redistribution* on *page 86* for licensing terms.)

If you created an application based on the One Touch for Windows: C/C++ Edition APIs that does not include an installer, your end users must install the One Touch for Windows: C/C++ Edition Runtime Environment to run your application.

To install the One Touch for Windows: C/C++ Edition RTE for 32-bit operating systems

1. In the RTE folder in the SDK software package, open the Setup.exe file.

2. Follow the installation instructions as they appear.

Table 2 identifies the files that are installed on your hard disk.

Table 2. One Touch for Windows: C/C++ Edition RTE installed files and folders, 32-bit installation

Folder	File	Description
<destination folder>\Bin	DPCOper2.dll DPDevice2.dll DPDevTS.dll DpHostW.exe DPmsg.dll DPMux.dll DpSvInfo2.dll DPTSCInt.dll DPCrStor.dll	DLLs and executable file used by the all of the One Touch for Windows APIs
<system folder>	DPFPApi.dll DpClback.dll dpHFtrEx.dll dpHMatch.dll DPFpUI.dll	DLLs used by all of the One Touch for Windows SDK APIs

To install the One Touch for Windows: C/C++ Edition RTE for 64-bit operating systems

1. In the RTE\x64 folder in the SDK software package, open the Setup.exe file.

2. Follow the installation instructions as they appear.

Table 3 identifies the files that are installed on your hard disk for 64-bit versions of the supported operating systems.

Table 3. One Touch for Windows: C/C++ Edition RTE installed files and folders, 64-bit installation

Folder	File	Description
<destination folder>\Bin	DPCOper2.dll DPDevice2.dll DPDevTS.dll DpHostW.exe DPMux.dll DpSvInfo2.dll DPTSCInt.dll DPCrStor.dll	DLLs and executable file used by the all of the One Touch for Windows APIs
<destination folder>\Bin\x64	DPmsg.dll	DLL used by the all of the One Touch for Windows APIs
<system folder>	DPFPApi.dll DpClback.dll dpHFtrEx.dll dpHMatch.dll DPFpUI.dll	32-bit DLLs used by all of the One Touch for Windows APIs
<system64 folder>	DPFPApi.dll DpClback.dll dpHFtrEx.dll dpHMatch.dll DPFpUI.dll	64-bit DLLs used by all of the One Touch for Windows APIs

Installing and Uninstalling the RTE Silently

The One Touch for Windows SDK software package contains a batch file, InstallOnly.bat, that you can use to silently install the RTE. In addition, you can modify the file to selectively install the various features of the RTE. Refer to the file for instructions.

The SDK software package also contains a file, UninstallOnly.bat, that you can use to silently uninstall the RTE.

Overview 4

This chapter introduces One Touch for Windows SDK: C/C++ Edition concepts and terminology. This chapter also includes typical workflow diagrams and explanations of the One Touch for Windows: C/C++ Edition API functions used to perform the tasks in the workflows. For additional information on fingerprint biometrics, refer to the "DigitalPersona White Paper: Guide to Fingerprint Recognition" included in the One Touch for Windows SDK software package "doc" folder as "Fingerprint Guide.pdf."

Biometric System

A *biometric system* is an automatic method of identifying a person based on the person's unique physical and/or behavioral traits, such as a fingerprint or an iris pattern, or a handwritten signature or voice. Biometric identifiers are

- Universal
- Distinctive
- Persistent (sufficiently unchangeable over time)
- Collectable

Biometric systems have become an essential component of effective person recognition solutions because biometric identifiers cannot be shared or misplaced and they naturally represent an individual's bodily identity. Substitute forms of identity, such as passwords (commonly used in logical access control) and identity cards (frequently used for physical access control), do not provide this level of authentication that strongly validates the link to the actual authorized user.

Fingerprint recognition is the most popular and mature biometric system used today. In addition to meeting the four criteria above, fingerprint recognition systems perform well (that is, they are accurate, fast, and robust), they are publicly acceptable, and they are hard to circumvent.

Fingerprint

A *fingerprint* is an impression of the ridges on the skin of a finger. A *fingerprint recognition system* uses the distinctive and persistent characteristics from the ridges, also referred to as *fingerprint features*, to distinguish one finger (or person) from another. The One Touch for Windows SDK: C/C++ Edition incorporates the *DigitalPersona Fingerprint Recognition Engine (Engine)*, which uses traditional as well as modern fingerprint recognition methodologies to convert these fingerprint features into a format that is compact, distinguishing, and persistent. The Engine then uses the converted, or extracted, fingerprint features in comparison and decision-making to provide reliable personal recognition.

Fingerprint Recognition

The DigitalPersona fingerprint recognition system uses the processes of fingerprint enrollment and fingerprint verification, which are illustrated in the block diagram in Figure 1 on *page 19*. Some of the tasks in these processes are done by the *fingerprint reader* and its driver; some are accomplished using One Touch for Windows: C/C++ Edition API functions, which use the Engine; and some are provided by your software application and/or hardware.

Fingerprint Enrollment

Fingerprint enrollment is the initial process of collecting *fingerprint data* from a person (*enrollee*) and storing the resulting data as a *fingerprint template* for later comparison. The following procedure describes typical fingerprint enrollment. (Steps preceded by an asterisk are not performed by the One Touch for Windows SDK: C/C++ Edition.)

1. *Obtain the enrollee's identifier (*Subject Identifier*).

2. Capture the enrollee's fingerprint using the fingerprint reader.

3. Extract the *fingerprint feature set* for the purpose of enrollment from the fingerprint sample.

4. Repeat steps 2 and 3 until you have enough fingerprint feature sets to create a fingerprint template.

5. Create a fingerprint template.

6. *Associate the fingerprint template with the enrollee through a Subject Identifier, such as a user name, email address, or employee number.

7. *Store the fingerprint template, along with the Subject Identifier, for later comparison.

   Fingerprint templates can be stored in any type of repository that you choose, such as a *fingerprint capture device*, a smart card, or a local or central database.

Fingerprint Verification

Fingerprint verification is the process of comparing the fingerprint data to the fingerprint template produced at enrollment and deciding if the two match. The following procedure describes typical fingerprint verification. (Steps preceded by an asterisk are not performed by the One Touch for Windows SDK: C/C++ Edition.)

1. *Obtain the Subject Identifier of the person to be verified.

2. Capture a fingerprint sample using the fingerprint reader.

3. Extract a fingerprint feature set for the purpose of verification from the fingerprint sample.

4. *Retrieve the fingerprint template associated with the Subject Identifier from your repository.

5. Perform a *one-to-one comparison* between the fingerprint feature set and the fingerprint template, and make a decision of *match* or *non-match*.

6. *Act on the decision accordingly, for example, unlock the door to a building for a match, or deny access to the building for a non-match.

Fingerprint Enrollment

| Get Subject Identifier |

| Capture fingerprints | → | Create fingerprint feature sets | → | Create template | → | Store template and Subject Identifier |

Fingerprint Verification

| Get Subject Identifier | → | Retrieve template and Subject Identifier |

| Capture fingerprint | → | Create fingerprint feature set | → | Compare and make decision | → | Use decision in application |

Legend

| | Performed by One Touch for Windows SDK: C/C++ |
| | Not performed by One Touch for Windows SDK: C/C++ |

Figure 1. DigitalPersona fingerprint recognition system

False Positives and False Negatives

Fingerprint recognition systems provide many security and convenience advantages over traditional methods of recognition. However, they are essentially pattern recognition systems that inherently occasionally make certain errors because no two impressions of the same finger are identical. During verification, sometimes a person who is legitimately enrolled is rejected by the system (a false negative decision), and sometimes a person who is not enrolled is accepted by the system (a false positive decision).

The proportion of false positive decisions is known as the *false accept rate (FAR)*, and the proportion of false negative decisions is known as the *false reject rate (FRR)*. In fingerprint recognition systems, the FAR and the FRR are traded off against each other, that is, the lower the FAR, the higher the FRR, and the higher the FAR, the lower the FRR.

A One Touch for Windows: C/C++ Edition API function enables you to set the value of the FAR, also referred to as the *security level*, to accommodate the needs of your application. In some applications, such as an access

control system to a highly confidential site or database, a lower FAR is required. In other applications, such as an entry system to an entertainment theme park, security (which reduces ticket fraud committed by a small fraction of patrons by sharing their entry tickets) may not be as significant as accessibility for all of the patrons, and it may be preferable to decrease the FRR at the expense of an increased FAR.

It is important to remember that the accuracy of the fingerprint recognition system is largely related to the quality of the fingerprint. Testing with sizable groups of people over an extended period has shown that a majority of people have feature-rich, high-quality fingerprints. These fingerprints will almost surely be recognized accurately by the DigitalPersona Fingerprint Recognition Engine and practically never be falsely accepted or falsely rejected. The DigitalPersona fingerprint recognition system is optimized to recognize fingerprints of poor quality. However, a very small number of people may have to try a second or even a third time to obtain an accurate reading. Their fingerprints may be difficult to verify because they are either worn from manual labor or have unreadable ridges. Instruction in the proper use of the fingerprint reader will help these people achieve the desired results.

Operations

Each time the user puts a finger on the fingerprint reader, fingerprint-related data is sent to the Engine. When the client application needs to perform some action requiring scanning a fingerprint, it should create an *operation*.

There is only one type of operation supported: Fingerprint sample acquisition. Right now the Engine supports only one type of fingerprint sample, which is a fingerprint image. During the creation of a fingerprint sample acquisition operation, the client application may specify its priority level, which can be low, normal, or high.

Note that no more than one client application may obtain the results of a single fingerprint scan.

It is possible to create and register any number of operations with normal priority, but no more than one operation for each of low and high priority at a time.

When the Engine is ready to dispatch the result of fingerprint scan, it processes operations using the following rules in the sequence shown.

1. If there is a high-priority operation registered, the result is dispatched to the process that owns the operation.

2. If there is no high-priority operation registered, the engine determines which process owns the topmost window. If there is a normal-priority operation owned by this process, it will receive the result.

3. If the above-mentioned steps do not allow the engine to dispatch the result, the process owning the low-priority operation (if registered) will receive the result.

If the result is still not dispatched, it is discarded.

Components of the SDK

The One Touch for Windows SDK: C/C++ Edition consists of the following two components:

- Device component

  The device component directs fingerprint reader (device) data and events to your application.

- Fingerprint recognition component

  The fingerprint recognition component performs fingerprint enrollment and verification and includes two modules: the fingerprint feature extraction module and the fingerprint comparison module.

Device Component

The device component workflow is shown below and is followed by explanations of the One Touch for Windows: C/C++ Edition Core API functions that are used to perform the tasks in the workflow.
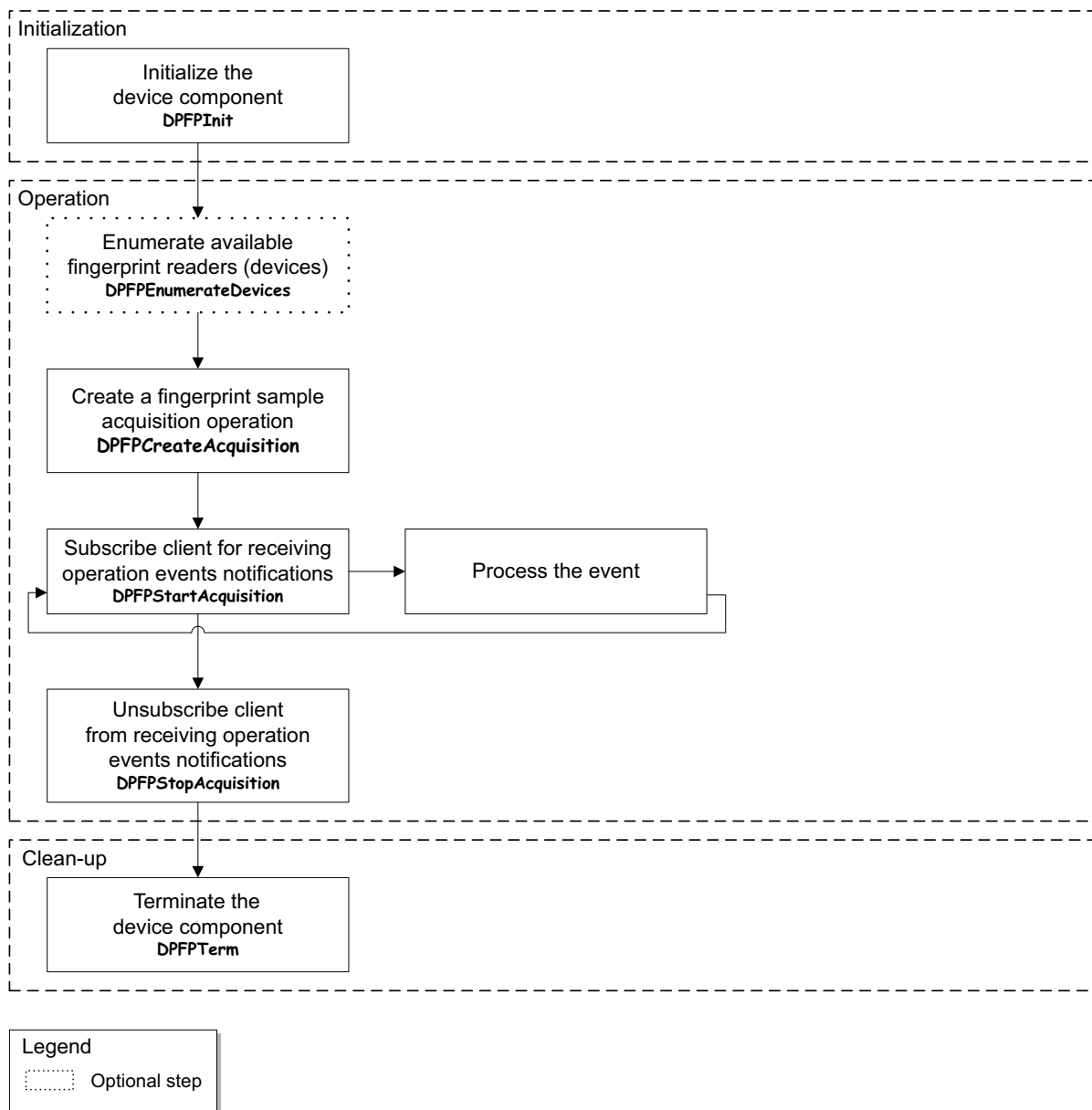
Figure 2. Device component workflow

Initialization

■ Initialize the device component by calling the **DPFPInit** function (*page 44*).

Operation

1. (Optional) If necessary, enumerate the available fingerprint readers (devices) connected to a computer by calling the **DPFPEnumerateDevices** function (*page 41*).

2. Create a fingerprint sample acquisition operation by calling the **DPFPCreateAcquisition** (*page 38*) and specifying the device's UID. You can also subscribe to all available fingerprint readers by passing the value **GUID_NULL**.

3. Subscribe the client application for receiving operation events notifications by calling **DPFPStartAcquisition** (*page 45*) and passing the operation handle.

4. Process the event.

5. Unsubscribe by calling the **DPFPStopAcquisition** function (*page 46*)

6. Release a subscribed fingerprint reader by calling the **DPFPDestroyAcquisition** function (*page 40*).

Clean-up

■ Terminate the device component when your application no longer requires access to any fingerprint readers by calling the **DPFPTerm** function (*page 46*).

Fingerprint Recognition Component

This section includes illustrations of typical fingerprint enrollment and verification workflows for the fingerprint recognition component and explanations of the One Touch for Windows: C/C++ Edition Core API functions used to perform the tasks in the workflows. Your application workflows may be different than those illustrated here. For example, you could choose to create fingerprint feature sets locally and then send them to a server for enrollment.

Fingerprint Enrollment

A typical fingerprint enrollment application workflow is represented below. Each figure is followed by explanations of the One Touch for Windows: C/C++ Edition Core API functions that are used to perform the tasks in that part of the workflow. Both the fingerprint feature extraction and the fingerprint comparison modules are used for performing enrollment.

Typical Fingerprint Enrollment Workflow
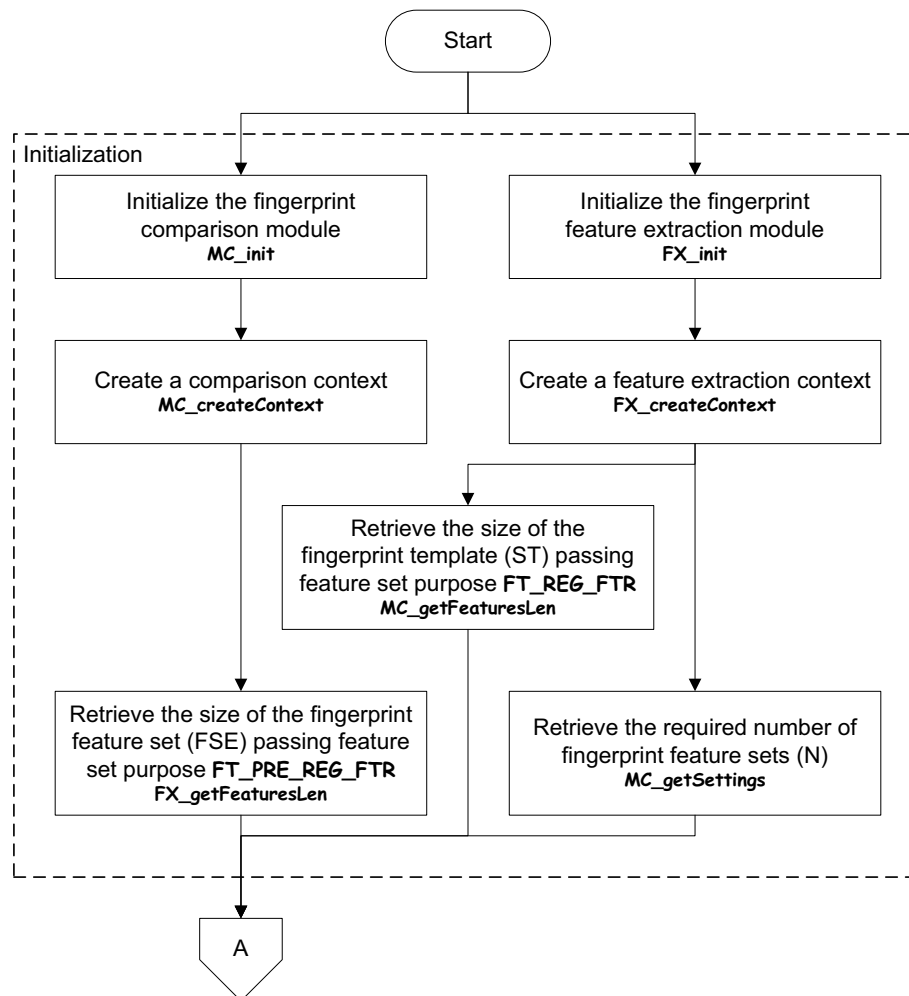
Figure 3. Typical fingerprint enrollment workflow: Initialization

Initialization Tasks

Steps 3 and 4 can be done before steps 1 and 2.

1. Initialize the fingerprint feature extraction module by calling the `FX_init` function (*page 47*).

2. Create a feature extraction *context* by calling the `FX_createContext` function (*page 48*)

3. Initialize the fingerprint comparison module by calling the `MC_init` function (*page 55*).

4. Create a comparison context by calling the `MC_createContext` function (*page 56*).

Steps 5 through 7 can be done in any order.

5. Retrieve the size of the fingerprint feature set (FSE) by calling the **FX_getFeaturesLen** function and passing feature set purpose **FT_PRE_REG_FTR** (*page 50*).

6. Retrieve the size of the fingerprint template (ST) by calling the **MC_getFeaturesLen** function and passing feature set purpose **FT_REG_FTR** (*page 59*).

7. Retrieve the number (N) of fingerprint feature sets required to create the fingerprint template by calling the **MC_getSettings** function (*page 56*).
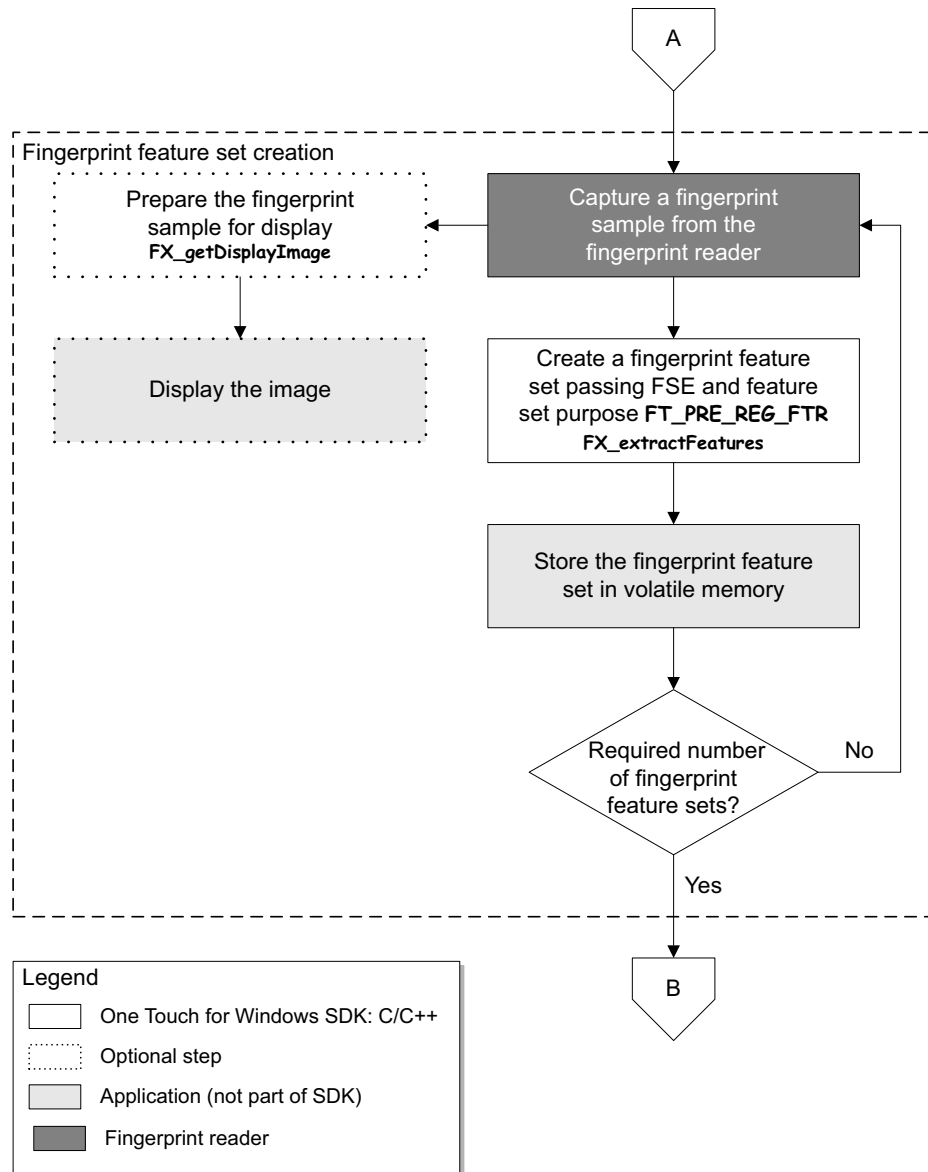
Figure 4. Typical fingerprint enrollment workflow: Fingerprint feature set creation

Fingerprint Feature Set Creation Tasks

Repeat the following required steps until you have created the number of fingerprint feature sets required to generate a fingerprint template. This number (N) was obtained when you called the **MC_getSettings** function during initialization (*page 56*). (Steps preceded by an asterisk are not accomplished using One Touch for Windows: C/C++ Edition Core API functions.)

1. *Capture a fingerprint image from the fingerprint reader.

2. Create a fingerprint feature set by calling the **FX_extractFeatures** function and passing FSE and feature set purpose **FT_PRE_REG_FTR** (*page 51*).

Steps 3 and 4 are optional.

3. Prepare the fingerprint image captured by the fingerprint reader for display by calling the **FX_getDisplayImage** function (*page 53*).

4. *Display the image.

5. *If the **FX_extractFeatures** function succeeds, store the resulting fingerprint feature set in volatile memory.

B

Fingerprint template creation

Create a fingerprint template passing ST and N number of fingerprint feature sets
MC_generateRegFeatures

Store the fingerprint template and associate it with a user ID

Clean-up

| Destroy the comparison context **MC_closeContext** | Destroy the feature extraction context **FX_closeContext** |

| Terminate the fingerprint comparison module **MC_terminate** | Terminate the fingerprint feature extraction module **FX_terminate** |

End

Legend

One Touch for Windows SDK: C/C++

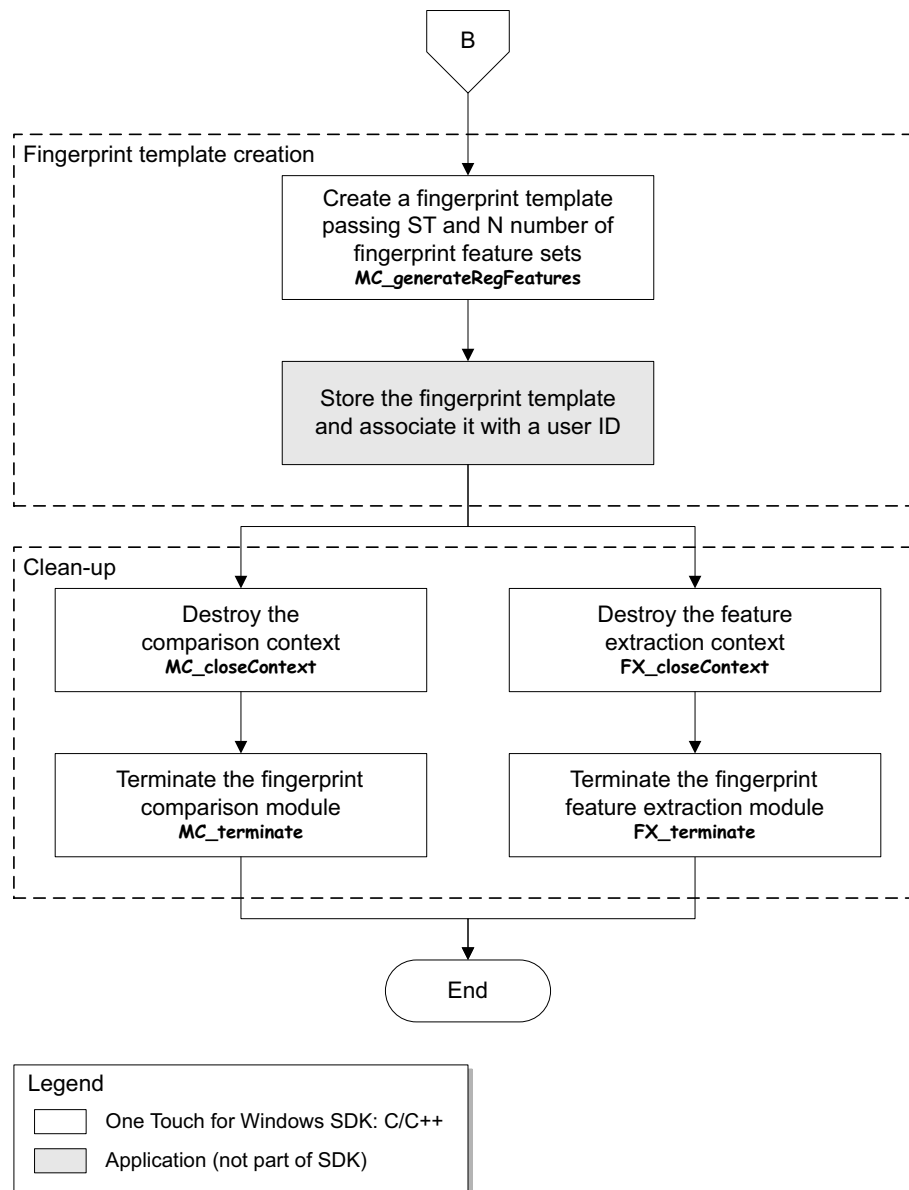Application (not part of SDK)

Figure 5. Typical fingerprint enrollment workflow: Fingerprint template creation and clean-up

Fingerprint Template Creation Tasks

1. Create a fingerprint template by calling the **`MC_generateRegFeatures`** function and passing ST and the N number of fingerprint features sets created previously and stored in volatile memory (*page 60*).

Step 2 is not accomplished using One Touch for Windows: C/C++ Edition Core API functions.

2. Store the fingerprint template in your repository and associate it with a user ID.

Clean-up Tasks

Steps 3 and 4 can be done before steps 1 and 2; however, during clean-up, you should always terminate modules in the reverse order of their initialization. In other words, if you initialize the fingerprint feature extraction module first, you should terminate that module last, and if you initialize the comparison module first, you should terminate that module last.

1. Destroy the comparison context by calling the **`MC_closeContext`** function (*page 57*)

2. Terminate the fingerprint comparison module by calling the **`MC_terminate`** function (*page 59*).

3. Destroy the feature extraction context by calling the **`FX_closeContext`** function (*page 49*)

4. Terminate the fingerprint feature extraction module by calling the **`FX_terminate`** function (*page 49*).

Fingerprint Verification

A typical fingerprint verification application workflow is represented in the following three illustrations. Each figure is followed by explanations of the One Touch for Windows: C/C++ Edition Core API functions that are used to perform the tasks in that part of the workflow. Both the fingerprint feature extraction and the fingerprint comparison modules are used for performing verification.

Typical Fingerprint Verification Workflow
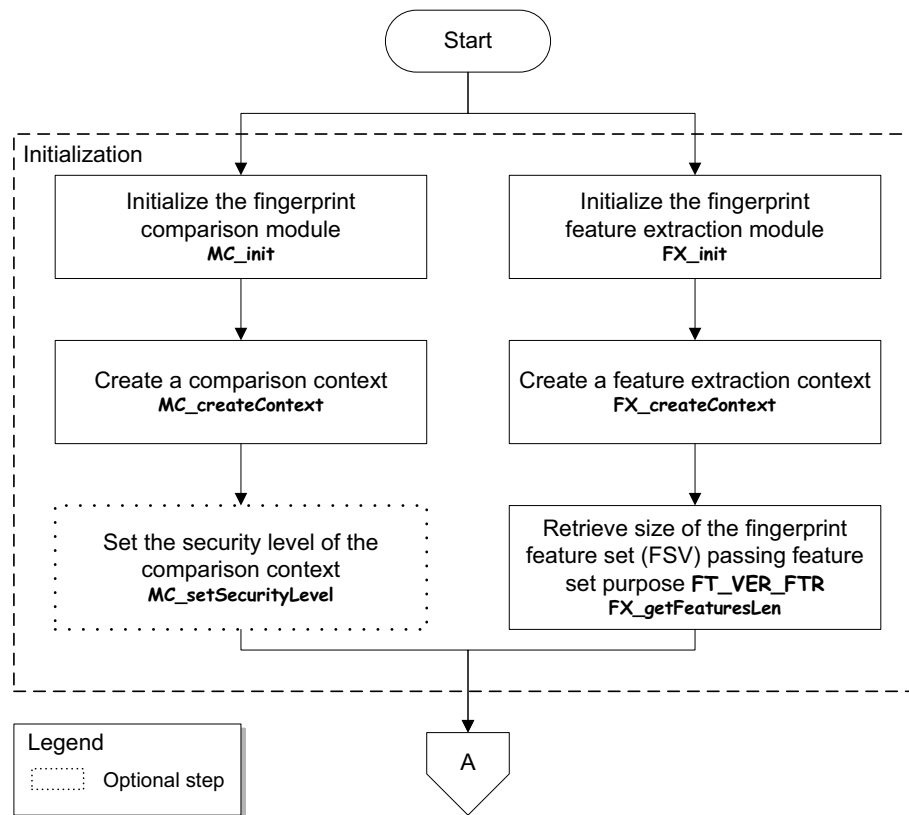
Figure 6. Typical fingerprint enrollment workflow: Initialization

Initialization Tasks

Steps 3 and 4 can be done before steps 1 and 2.

1. Initialize the fingerprint feature extraction module by calling the `FX_init` function (*page 47*).

2. Create a feature extraction context by calling the `FX_createContext` function (*page 48*)

3. Initialize the fingerprint comparison module by calling the `MC_init` function (*page 55*).

4. Create a comparison context by calling the `MC_createContext` function (*page 56*).

5. Optionally, set the security level of the comparison context by calling the `MC_setSecurityLevel` function (*page 57*). If you do not call this function, the default security level will be used.

6. Retrieve the size of the fingerprint feature set (FSV) by calling the `FX_getFeaturesLen` function and passing feature set purpose `FT_VER_FTR` (*page 50*).

```
                                    ┌────────┐
                                    │   A    │
                                    └────┬───┘
                                         │
  ┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐
   Fingerprint feature set creation
  │                                                             │
      ┌──────────────────────────────┐
  │   │      Acquire a user ID        │────────────┐           │
      └──────────────────────────────┘            │
  │                                                ▼           │
      ┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐         ┌──────────────────────┐
  │     Prepare the fingerprint         │   Capture a fingerprint│
        sample for display    ◄────────┤      sample from the  │
  │     FX_getDisplayImage            │ │    fingerprint reader │
      └ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘         └──────────┬───────────┘ │
  │            │                                  │
               ▼                                  ▼             │
  │   ┌ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┐         ┌──────────────────────┐
                                     │  Create a fingerprint │  │
  │     Display the image   │        │  feature set passing  │
                                     │  FSV and feature set  │  │
  │   └ ─ ─ ─ ─ ─ ─ ─ ─ ─ ┘         │  purpose FT_VER_FTR   │
                                     │    FX_extractFeatures │  │
  │                                  └──────────┬───────────┘
                                                │                │
  │                                  ┌──────────▼───────────┐
                                     │ Store the fingerprint│   │
  │                                  │ feature set in        │
                                     │ volatile memory       │   │
  │                                  └──────────┬───────────┘
                                                │                │
  └ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─┼─ ─ ─ ─ ─ ─ ─ ┘
                                                ▼
                                          ┌────────┐
  ┌──────────────────────────────┐        │   B    │
  │ Legend                        │        └────────┘
  │  ┌───┐  One Touch for Windows │
  │  └───┘  SDK: C/C++            │
  │  ┌┄┄┄┐  Optional step         │
  │  └┄┄┄┘                        │
  │  ┌───┐  Application (not part │
  │  └───┘  of SDK)               │
  │  ┌───┐  Fingerprint reader    │
  │  └───┘                        │
  └──────────────────────────────┘
```
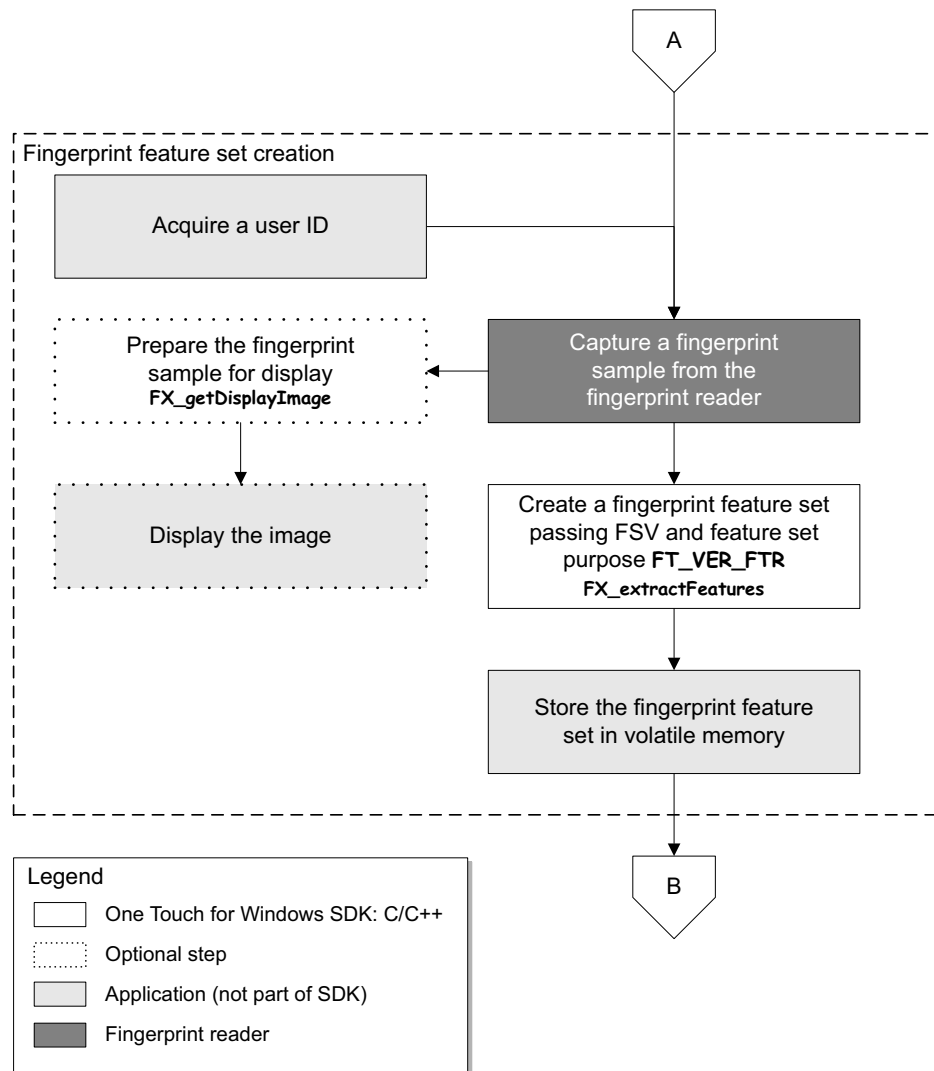

Figure 7. Typical fingerprint verification workflow: Fingerprint feature set creation

Fingerprint Feature Set Creation Tasks

Steps preceded by an asterisk are not accomplished using One Touch for Windows: C/C++ Edition Core API functions.

1. *Acquire the user ID that was used to associate the fingerprint template with the person to be verified.

2. *Capture a fingerprint image from the person via the fingerprint reader.

3. Create a fingerprint feature set by calling the **FX_extractFeatures** function and passing FSV and feature set purpose **FT_VER_FTR** (*page 51*).

Steps 4 and 5 are optional.

4. Prepare the fingerprint image captured by the fingerprint reader for display by calling the **FX_getDisplayImage** function (*page 53*).

5. *Display the image.

6. *If the **FX_extractFeatures** function succeeds, store the resulting fingerprint feature set in volatile memory.
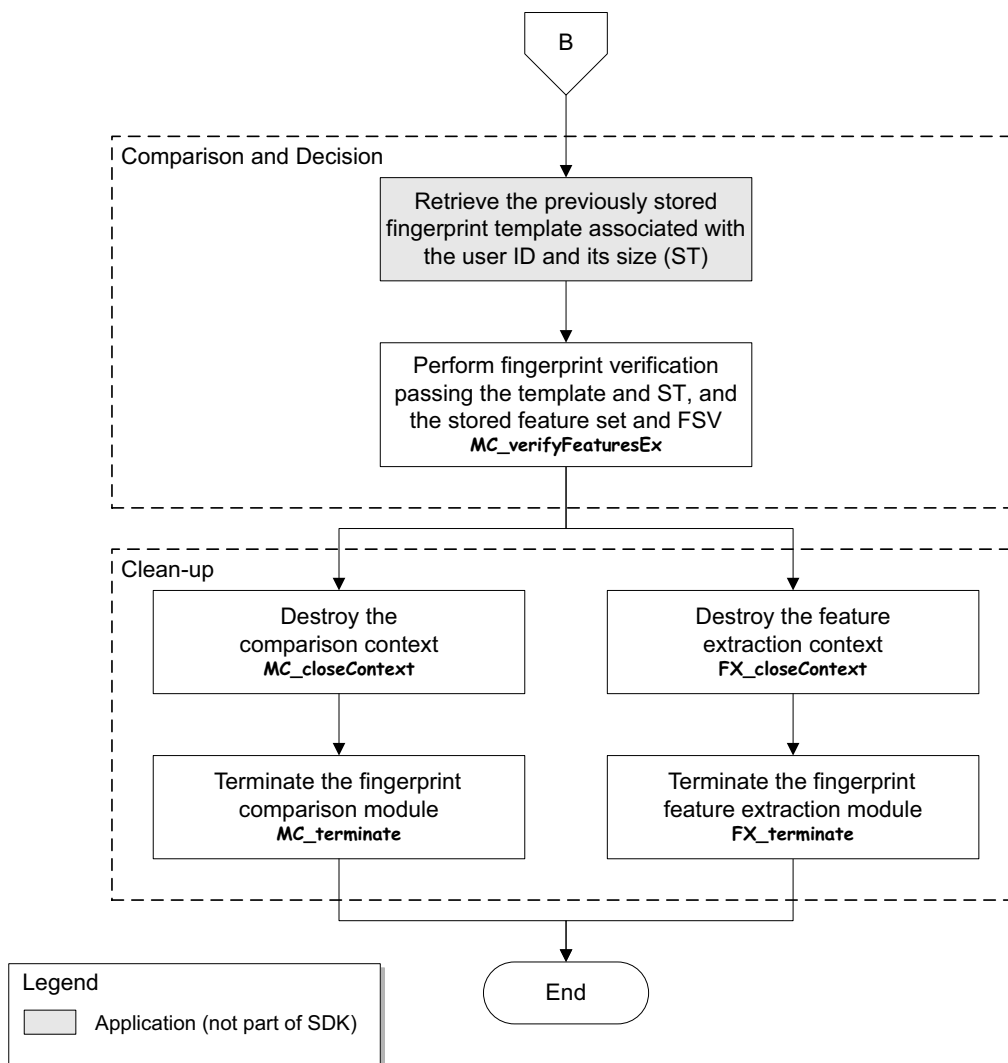
Figure 8. Typical fingerprint verification workflow: Comparison and decision and clean-up

Comparison-and-Decision Tasks

Step 1 is not accomplished using One Touch for Windows: C/C++ Edition Core API functions.

1. *Retrieve the fingerprint template associated with the user ID and size ST from your repository.

2. Perform fingerprint verification by calling the **`MC_verifyFeaturesEx`** function and passing the stored fingerprint feature set together with FSV, and the fingerprint template together with ST (*page 62*).

Clean-up Tasks

Steps 3 and 4 can be done before steps 1 and 2; however, during clean-up, you should always terminate modules in the reverse order of their initialization. In other words, if you initialize the fingerprint feature extraction module first, you should terminate that module last, and if you initialize the comparison module first, you should terminate that module last.

1. Destroy the comparison context by calling the **`MC_closeContext`** function (*page 57*)

2. Terminate the fingerprint comparison module by calling the **`MC_terminate`** function (*page 59*).

3. Destroy the feature extraction context by calling the **`FX_closeContext`** function (*page 49*)

4. Terminate the fingerprint feature extraction module by calling the **`FX_terminate`** function (*page 49*).

This chapter provides a reference to the One Touch for Windows: C/C++ Edition Core API, including information about its

- *Functions* on *page 35*

- *Data Structures* on *page 64*

- *Enumerations* on *page 68*

- *Type Definitions and Constants* on *page 74*

The next chapter, *User Interface API Reference*, describes the DPUIAPI wrapper that simplifies access to the entire functionality available in the Core API described in this chapter. The wrapper provides a premade user interface that handles device component, fingerprint enrollment, and fingerprint verification tasks through only two functions and two callbacks.

This chapter defines the One Touch for Windows: C/C++ Edition Core API functions. Use the three categorized lists in this section to quickly find the functions contained in the following pages by function name or by description.

Functions

The functions are arranged for convenience into these three categories.

- Device functions - are used to communicate with the U.are.U fingerprint reader.

- Extraction functions - are used for performing feature extraction, which is the system function that is applied to a fingerprint sample to compute repeatable and distinctive information to be used for fingerprint verification or fingerprint enrollment.

- Matching - compares a fingerprint template and a feature set and calculates a score that indicates how likely it is that they come from the same finger.

Device Functions List

Function	Page	Description
DPFPBufferFree	38	Frees memory previously allocated by a DPFP function call.
DPFPCreateAcquisition	38	Creates a fingerprint sample acquisition operation.
DPFPDestroyAcquisition	40	Destroys the operation previously created by `DPFPCreateAcquisition` and deallocates all resources associated with that operation.
DPFPEnumerateDevices	41	Enumerates fingerprint readers (devices) connected to the computer.
DPFPGetDeviceInfo	41	Retrieves information about a specified fingerprint reader.

Function	Page	Description
DPFPSetDeviceParameter	*42*	Sets a specified value to a dedicated storage location on a U.are.U fingerprint reader.
DPFPGetDevice Parameter	*43*	Gets the value contained in a dedicated storage location on a U.are.U fingerprint reader.
DPFPGetVersion	*44*	Gets the API version information.
DPFPInit	*44*	Initializes the device component and allocates necessary resources.
DPFPStartAcquisition	*45*	Subscribes the client application for receiving operation events notifications.
DPFPStopAcquisition	*46*	Unsubscribes the client application from receiving operation events
DPFPTerm	*46*	Terminates the device component and deallocates resources.

Extraction Functions List

Function	Page	Description
- Module Initialization, Settings and Termination -		
FX_init	*47*	Initializes the dpFtrEx module.
FX_getVersionInfo	*48*	Returns the software version of the module.
FX_createContext	*48*	Creates a feature extraction context.
FX_closeContext	*49*	Closes a feature extraction context.
FX_terminate	*49*	Closes the dpFtrEx module.
- Feature Manipulation -		
FX_getFeaturesLen	*50*	Returns the length of the features of the specified type.
FX_extractFeatures	*51*	Extracts the features from a given scan.
FX_getDisplayImage	*53*	Returns an image prepared for display.

Matching Functions List

Function	Page	Description
- Module Initialization, Settings, and Termination -		
MC_init	*55*	Initializes the dpMatch module.
MC_getVersionInfo	*55*	Returns the software version of the module.

Function	Page	Description
MC_getSettings	*56*	Returns the number of fingerprint feature sets required.
MC_createContext	*56*	Creates a matching context.
MC_closeContext	*57*	Closes a matching context.
MC_getSecurityLevel	*57*	Returns the current security level.
MC_setSecurityLevel	*58*	Sets the security level for a matching context.
MC_terminate	*59*	Closes the dpMatch module.
- Fingerprint Enrollment and Verification -		
MC_getFeaturesLen	*59*	Returns the minimum and recommended length of the features.
MC_generateRegFeatures	*60*	Creates a fingerprint template.
MC_verifyFeaturesEx	*62*	Compares a fingerprint template with a fingerprint feature set.

Device Functions Reference

Device functions are used to communicate with the U.are.U fingerprint reader, and are listed in the pages that follow.

DPFPBufferFree

Frees memory previously allocated by a DPFP function call.

Syntax

```
void DPFPBufferFree (PVOID p);
```

Parameter Names

`p`	[in] The memory area to be freed.

Library

DPFPApi.lib

DPFPCreateAcquisition

Creates a fingerprint sample acquisition operation.

Syntax

```
HRESULT DPFPCreateAcquisition(
   DP_ACQUISITION_PRIORITY eAcquisitionPriority,
   REFGUID DevUID,
   ULONG uSampleType,
   HWND hWnd,
   ULONG uMsg,
   HDPOPERATION * phOperation
);
```

Parameters

eAcquisitionPriority	[in] Acquisition priority needed. Must be one of the following values:	
	DP_PRIORITY_HIGH	The process subscribing with this priority acquires device events exclusively.
		The process subscribing with this priority must have administrative privileges or run under Local SYSTEM account.
		Only one subscriber with this priority is allowed.
	DP_PRIORITY_NORMAL	The process subscribing with this priority acquires device events only if it runs as a foreground process.
		Multiple subscribers with this priority are allowed.
	DP_PRIORITY_LOW	The process subscribing with this priority acquires device events only if there are no subscribers with higher priority.
		Only one subscriber with this priority is allowed.
DevUID	[in] Fingerprint Reader serial number. Can be **GUID_NULL** if any reader can be used.	
uSampleType	[in] Type of fingerprint sample needed. This parameter must be **DP_SAMPLE_TYPE_IMAGE**.	
hWnd	[in] Handle of the window to be notified of operation events.	
uMsg	[in] Window message to be sent as an event notification.	
phOperation	[out] Pointer to operation handle to be filled if operation was created successfully.	

Return Values

S_OK	Operation was successfully created.
E_ACCESSDENIED	Application attempted to subscribe to device events using **DP_PRIORITY_HIGH** but the priority does not have adequate rights to do so. Only members of built-in Administrators group and Local SYSTEM account have the right to subscribe with **DP_PRIORITY_HIGH** priority. If DigitalPersona Pro is also installed on the same computer an administrator needs to allow use of this function by enabling "Allow Fingerprint Data Redirection" in the governing GPO. The **E_ACCESSDENIED** error will be returned otherwise.
E_INVALIDARG	For subscribers with **DP_PRIORITY_HIGH** and **DP_PRIORITY_LOW** priorities, this error indicates that there is another application which has already subscribed to device events with the same priority.

Remarks

In order to free memory allocated for the operation created, the client application must call **DPFPDestroyAcquisition** for the handle returned in **phOperation**.

Library

DPFPApi.lib

DPFPDestroyAcquisition

Destroys the operation previously created by **DPFPCreateAcquisition** and deallocates all resources associated with that operation.

Syntax

```
HRESULT DPFPDestroyAcquisition (HDPOPERATION hOperation);
```

Parameter Names

hOperation	[in] Handle to operation that is to be destroyed.

Return Values

S_OK	Operation was successfully destroyed.

Library

DPFPApi.lib

DPFPEnumerateDevices

Enumerates the device UIDs of available fingerprint readers (devices) connected to this computer.

Syntax

```
HRESULT DPFPEnumerateDevices (
    ULONG * puDevCount,
    GUID ** ppDevUID
);
```

Parameters

puDevCount	[out] Number of readers available. If no readers are found, this number is **0**.
puDevUID	[out] Pointer to be filled with the pointer to the array of device UIDs for available fingerprint readers. If **NULL**, only the number of available readers will be returned.

Return Values

S_OK	Function was successful.

Remarks

Caller must release returned memory by calling **DPFPBufferFree**.

Library

DPFPApi.lib

DPFPGetDeviceInfo

Retrieves information about a particular reader.

Syntax

```
HRESULT DPFPGetDeviceInfo (
    REFGUID DevUID,
    DP_DEVICE_INFO * pDevInfo
);
```

Parameters

DevUID	[in] Pointer to the UID of the fingerprint reader to retrieve information about.
pDevInfo	[in, out] Pointer to **DP_DEVICE_INFO** structure receiving information about the specified reader.

Return Values

S_OK	Information was retrieved.

Library

DPFPApi.lib

DPFPSetDeviceParameter

Writes a value (pData) to a dedicated storage location on a U.R.U fingerprint reader with the specified serial number.

Syntax

```
HRESULT DPFPSetDeviceParameter(
   REFGUID DevUID,
   unsigned long ulParamID,
   const DATA_BLOB* pData
);
```

Parameters

DevUID	[in] Fingerprint reader serial number. Can be GUID_NULL if any reader can be used.
ulParamID	Target Parameter ID, as follows: **FT_SET_CLIENT_PRIVATE_KEY** Writes a hashed private key into a fingerprint reader's persistant storage. This feature is supported on DigitalPersona's 4000B and later readers. The source data to hash is supplied through pData. The reader needs to be recycled (disconnected and reconnected) before the key can be retrieved through DPFPGetDeviceParameter.
pData	Parameter value to set.

Return Values

S_OK	Parameter was set.

Library

DPFPApi.lib

DPFPGetDeviceParameter

Retrieves a parameter value from a specified reader.

Syntax

```
HRESULT DPFPGetDeviceParameter,
    REFGUID DevUID,
    unsigned long ulParamID,
    const DATA_BLOB* pData
);
```

Parameters

`DevUID`	[in] Fingerprint reader serial number. Can be GUID_NULL if any reader can be used.
`ulParamID`	Target Parameter ID, as follows: **`FT_GET_CLIENT_PRIVATE_KEY`** Reads a private key hash stored within a fingerprint reader's persistant storage. This feature is supported on DigitalPersona's 4000B and later readers. The reader needs to be recycled (disconnected and reconnected) after invoking DPFPSetDeviceParameter, before the key can be retrieved. pData is required in order to allocate 16 bytes of storage when the function is used to retrieve this parameter.
`pData`	[out] Parameter value to retrieve. The caller has to allocate, load, and destroy DATA_BLOB members.

Return Values

`S_OK`	Parameter retrieved.

Library

DPFPApi.lib

DPFPGetVersion

Gets the API version information.

Syntax

```
HRESULT DPFPGetVersion (
    DP_PRODUCT_VERSION * pVersion
);
```

Parameters

`pVersion`	[out] Pointer to the structure to be filled.

Return Values

`S_OK`	Function was successful.

Library

DPFPApi.lib

DPFPInit

Allocates and initializes necessary resources. It MUST be called before any other DPFPApi calls except for `DPFPBufferFree`.

Syntax

```
HRESULT DPFPInit ();
```

Return Values

`S_OK`	Initialization successful.
`S_FALSE`	Library is already initialized.
`0x800706B3`	The RPC server is not listening, which means that the Biometric Authentication Service has not been started.

Remarks

Every successful (that is, `FAILED() == FALSE`) call of `DPFPInit` must have a corresponding call of the `DPFPTerm` function.

Library

DPFPApi.lib

DPFPStartAcquisition

Subscribes the client application for receiving operation events notifications.

Syntax

```
HRESULT DPFPStartAcquisition (
   HDPOPERATION hOperation
);
```

Parameters

`hOperation`	[in] Operation handle.

Return Values

`S_OK`	If subscription is successful.

Remarks

Each process can have no more than one active subscription for each operation priority level.

Library

DPFPApi.lib

DPFPStopAcquisition

Unsubscribes the client application from receiving operation events notifications.

Syntax

```
HRESULT DPFPStopAcquisition (
    HDPOPERATION hOperation
);
```

Parameters

`hOperation`	[in] Operation handle.

Return Values

`S_OK`	If unsubscription is successful.

Library

DPFPApi.lib

DPFPTerm

Deallocates resources allocated by `DPFPInit`.

Syntax

```
Void DPFPTerm ();
```

Library

DPFPApi.lib

Extraction Functions Reference

The dpFtrEx module contains code for performing feature extraction, which is the system function that is applied to a fingerprint sample to compute repeatable and distinctive information to be used for fingerprint verification or fingerprint enrollment.

The feature extraction modules maintain one or more contexts for each caller. A context can be created by calling **FX_createContext**, and is released with **FX_closeContext**.

Extraction functions are used to create feature extraction contexts, extract features, and prepare an image for display.

FX_extractFeatures is the function that extracts the features from the image, which is passed as one of the arguments. A handle to the context has to be passed to **FX_extractFeatures** and **FX_getDisplayImage**.

FX_init

Initializes the fingerprint extraction module. It reads various internal settings from the registry, initializes the **MC_SETTINGS** structure and initializes the lookup tables used for matching.

This function must be called before any other function in the module is called.

Syntax

```
FX_DLL_INTERFACE FT_RETCODE fx_init(void);
```

Return Values

FT_OK	Initialization successful.
FT_ERR_NO_MEMORY	There was not enough memory to initialize the feature extraction module.
FT_ERR_BAD_INI_SETTING	Initialization settings are corrupted.

Library

dpHFtrEx.lib

FX_getVersionInfo

Retrieves the software version of the feature extraction module in the structure of type **FT_VERSION_INFO**.

Syntax

```
FX_DLL_INTERFACE void FX_getVersionInfo(
  OUT FT_VERSION_INFO_PT fxModuleVersionPt
);
```

Parameters

fxModuleVersionPt	[out] Pointer to the buffer containing the software version of the fingerprint feature extraction module.

Return Values

FT_OK	Function successful.

Library

dpHFtrEx.lib

FX_createContext

Creates a feature extraction context. If this function succeeds, it returns the handle to the context that is created. All of the operations in this context require this handle.

Syntax

```
FX_DLL_INTERFACE FT_RETCODE FX_createContext(
  OUT FT_HANDLE *fxContext
);
```

Parameters

fxContext	[out] Pointer to the memory location where the context handle will be placed.

Return Values

FT_OK	The function succeeded.
FT_ERR_NO_INIT	**FX_init** has not yet been successfully called. The feature extraction module has not been initialized.
FT_ERR_INVALID_CONTEXT	There is not enough memory to create a context.

Library

dpHFtrEx.lib

FX_closeContext

Destroys a feature extraction context created by **FX_createContext** and releases the allocated resources.

Syntax

```
FX_DLL_INTERFACE FT_RETCODE FX_closeContext(
  IN FT_HANDLE fxContext
);
```

Parameters

fxContext	[in] Pointer to the context handle of the context to be closed.

Return Values

FT_OK	The function succeeded.
FT_ERR_NO_INIT	**FX_init** not yet been successfully called. The feature extraction module has not been initialized.
FT_ERR_INVALID_CONTEXT	The given feature extraction context is not valid.

Library

dpHFtrEx.lib

FX_terminate

Terminates the fingerprint extraction module and releases all resources associated with it.

Syntax

```
FX_DLL_INTERFACE FT_RETCODE FX_terminate (void);
```

Return Values

FT_OK	The function succeeded.
FT_WRN_NO_INIT	The feature extraction module has not been initialized.

Library

dplibrary.dll

FX_getFeaturesLen

Retrieves the size of the buffer for the fingerprint feature set. This function returns either the minimum or the recommended size that provides the best recognition accuracy, or both.

Syntax

```
FX_DLL_INTERFACE FT_RETCODE FX_getFeaturesLen(
   IN FT_FTR_TYPE featureSetPurpose,
   OUT int* recommendedFeatureSetSize,
   OUT int* minimumFeatureSetSize
);
```

Parameters

`featureSetPurpose`	[in] Feature set purpose. Specifies the purpose for which the fingerprint feature set is to be created. For a fingerprint feature set to be used for enrollment, use the value `FT_PRE_REG_FTR`; for verification, use `FT_VER_FTR`. `FT_REG_FTR` is not a valid value for this function.
`recommendedFeatureSetSize`	Pointer to the memory receiving the size of the buffer for the fingerprint feature set recommended for best recognition accuracy, or `NULL`.
	If `NULL` is passed, `minimumFeatureSetSize` must not be `NULL`.
`minimumFeatureSetSize`	Pointer to the memory receiving the minimum size of the buffer for the fingerprint feature set, or `NULL`.
	If `NULL` is passed, `recommendedFeatureSet` must not be `NULL`.

Return Values

`FT_OK`	The function succeeded.
`FT_ERR_NO_INIT`	The feature extraction module has not been initialized.
`FT_ERR_INVALID_PARAM`	The `featureSetPurpose` parameter is not valid.

Library

dpHFtrEx.lib

FX_extractFeatures

Creates a fingerprint feature set by applying *fingerprint feature extraction* to the fingerprint image obtained from the fingerprint reader to compute repeatable and distinctive information. Depending on the specified feature set purpose, this information can be used for either fingerprint enrollment or verification.

Syntax

```
FX_DLL_INTERFACE FT_RETCODE FX_extractFeatures(
   IN FT_HANDLE fxContext,
   IN int imageSize,
   IN const FT_IMAGE_PTC imagePt,
   IN FT_FTR_TYPE featureSetPurpose,
   IN int featureSetSize,
   OUT FT_BYTE* featureSet,
   OUT FT_IMG_QUALITY_PT imageQualityPt,
   OUT FT_FTR_QUALITY_PT featuresQualityPt,
   OUT FT_BOOL* featureSetCreated
);
```

Parameter Names

`fxContext`	[in] Handle to the feature extraction context
`imageSize`	[in] Size in bytes of the image obtained from the fingerprint reader.
`imagePt`	[in] Pointer to the buffer that contains the fingerprint image obtained from the fingerprint reader
`featureSetPurpose`	[in] Feature set purpose. Specifies the purpose for which the fingerprint feature set is to be created. For a fingerprint feature set to be used for enrollment, use the value `FT_PRE_REG_FTR`; for verification, use `FT_VER_FTR`. `FT_REG_FTR` is not a valid value for this function.
`featureSetSize`	[in] Fingerprint feature set size. This parameter is the size, in bytes, of the fingerprint feature set. Use the `FX_getFeaturesLen` function (*page 50*) to obtain information about which fingerprint feature set size to use.
`featureSet`	[out] Pointer to the buffer location receiving the fingerprint feature set

imageQualityPt	[out] Pointer to the buffer containing information about the quality of the fingerprint image. Image quality is represented by one of the following values:
	FT_GOOD_IMG. The fingerprint image quality is good.
	FT_IMG_TOO_LIGHT. The fingerprint image is too light.
	FT_IMG_TOO_DARK. The fingerprint image is too dark.
	FT_IMG_TOO_NOISY. The fingerprint image is too blurred.
	FT_LOW_CONTRAST. The fingerprint image contrast is too low.
	FT_UNKNOWN_IMG_QUALITY. The fingerprint image quality is undetermined.
featuresQualityPt	[out] Pointer to the buffer containing information about the quality of the fingerprint features. If the fingerprint image quality (**imageQualityPt**) is not equal to the value **FT_GOOD_IMG**, extraction is not attempted, and the parameter is set to F**T_UNKNOWN_FTR_QUALITY**.
	Fingerprint features quality is represented by one of the following values:
	FT_GOOD_FTR. The fingerprint features quality is good.
	FT_NOT_ENOUGH_FTR. There are not enough fingerprint features.
	FT_NO_CENTRAL_REGION. The fingerprint image does not contain the central portion of the finger.
	FT_AREA_TOO_SMALL. The fingerprint image area is too small.
	FT_UNKNOWN_FTR_QUALITY. Quality cannot be determined.
featureSetCreated	[out] Pointer to the memory receiving the value of whether the fingerprint feature set is created. If the value of this parameter is **FT_TRUE**, the fingerprint feature set was written to **featureSet**. If the value is **FT_FALSE**, a fingerprint feature set was not created.

Return Values

FT_OK	The function succeeded.
FT_ERR_NO_INIT	The fingerprint feature extraction module is not initialized.
FT_ERR_INVALID_CONTEXT	The given feature extraction context is not valid.
FT_ERR_INVALID_PARAM	One or more parameters are not valid.
FT_ERR_NO_MEMORY	Not enough memory to perform fingerprint feature extraction.
FT_ERR_UNKNOWN_DEVICE	The fingerprint reader is not supported.

Library

dpHFtrEx.lib

FX_getDisplayImage

Prepares the fingerprint image obtained from the fingerprint reader for display. This may involve resizing, changing the number of grayscale intensity levels, rotating, and otherwise processing the fingerprint image to ensure that it displays well. The fingerprint image passed to the **FX_getDisplayImage** function is the same fingerprint image used by the **FX_extractFeatures** function (*page 51*).

Syntax

```
FX_DLL_INTERFACE FT_RETCODE FX_getDisplayImage(
   IN FT_HANDLE fxContext,
   IN const FT_IMAGE_PTC imagePt,
   IN const FT_IMAGE_SIZE_PT pImageSize,
   IN const FT_BOOL imageRotation,
   IN const int numIntensityLevels,
   OUT FT_IMAGE_PT pImageBuffer
);
```

Parameter Names

fxContext	[in] Handle to the feature extraction context
imagePt	[in] Pointer to the buffer containing the fingerprint image obtained from the fingerprint reader
pImageSize	[in] Pointer to the buffer containing the requested dimensions (width and height) of the fingerprint image
imageRotation	[in] Indicates whether the fingerprint image is to be rotated. If the value of this parameter is equal to **FT_TRUE**, the fingerprint image is rotated. If the value is **FT_FALSE**, the fingerprint image is not rotated.
numIntensityLevels	[in] Requested number of grayscale intensity levels. Valid values are integers between 1 and 256.
pImageBuffer	[out] Pointer to the buffer which will be filled with display image bytes. Buffer must be large enough to hold the image information that will be returned, i.e. width times height of the image.

Return Values

FT_OK	The function succeeded.
FT_ERR_NO_INIT	The fingerprint feature extraction module is not initialized.
FT_ERR_INVALID_CONTEXT	The given feature extraction context is not valid.
FT_ERR_INVALID_PARAM	One or more parameters are not valid.
FT_ERR_NO_MEMORY	There is not enough memory to perform the function.
FT_ERR_UNKNOWN_DEVICE	The fingerprint reader is not supported.

Library

dpHFtrEx.lib

Matching Functions Reference

The dpMatch module contains code that compares a fingerprint template and a fingerprint feature set and calculates a score that indicates how likely it is that they come from the same finger.

The dpMatch module has a structure of type **MC_SETTINGS**, which is initialized by **MC_init**.

Most of the functions must be called in a particular context, which is specified by passing a context handle as the first argument.

MC_init

Initializes the fingerprint comparison module. This function must be called before any other functions in the module are called.

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_init(void);
```

Return Values

FT_OK	The function succeeded.
FT_ERR_BAD_INI_SETTING	Initialization settings are corrupted.
FT_ERR_NO_MEMORY	There is not enough memory to initialize the fingerprint comparison module.

Library

dpHMatch.lib

MC_getVersionInfo

Retrieves the software version information of the fingerprint comparison module.

Syntax

```
MC_DLL_INTERFACE void MC_getVersionInfo(
   OUT FT_VERSION_INFO_PT mcModuleVersionPt
);
```

Parameters

mcModuleVersionPt	[out] Pointer to software version of the fingerprint comparison module

Return Values

FT_OK	Function was completed successfully.

Library

dpHMatch.lib

MC_getSettings

Retrieves the current fingerprint comparison module settings in the structure of type **MC_SETTINGS**. This function provides the number of fingerprint feature sets required for the purpose of fingerprint enrollment. This setting is read-only.

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_getSettings(
   OUT MC_SETTINGS_PT mcSettingsPt
);
```

Parameters

mcSettingsPt	[out] Pointer to the structure of the fingerprint comparison module settings

Return Values

FT_OK	Function was completed successfully.

Library

dpHMatch.lib

MC_createContext

Creates a context for the fingerprint comparison module. If this function succeeds, it returns the handle to the context that is created. All of the operations in this context require this handle.

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_createContext(
   OUT FT_HANDLE* mcContext
);
```

Parameters

mcContext	[out] Pointer to the memory receiving the handle to the comparison context

Return Values

FT_OK	Function was completed successfully.

Library

dpHMatch.lib

MC_closeContext

Destroys a comparison context and releases the resources associated with it.

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_closeContext(
   IN FT_HANDLE mcContext
);
```

Parameters

mcContext	[in] Handle to the comparison module context

Return Values

FT_OK	Function was completed successfully.

Library

dpHMatch.lib

MC_getSecurityLevel

Retrieves the current security level of the specified comparison context in terms of the false accept rate (FAR).

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_getSecurityLevel(
   IN FT_HANDLE mcContext,
   OUT FT_FA_RATE* targetFar
);
```

Parameters

mcContext	[in] Handle to the comparison context
targetFar	[out] Pointer to the memory receiving the target FAR for the comparison context

Return Values

FT_OK	Function was completed successfully.

Library

dpHMatch.lib

MC_setSecurityLevel

Sets the security level of a comparison context by specifying a target false accept rate (FAR). The lower the value of the FAR, the higher the security level and the higher the target false reject rate (FRR). (See *False Positives and False Negatives* on *page 19* for more information about FAR and FRR.)

IMPORTANT: This function is to be used for comparison contexts only. *Do not* specify a security level for a feature extraction context.

IMPORTANT: Although the default value of **MED_SEC_FA_RATE** is adequate for most applications, you might require a lower or higher value to meet your needs. If you decide to use a value other than the default, be sure that you understand the consequences of doing so. Refer to Appendix A on *page 91* for more information about setting the value of the FAR.

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_setSecurityLevel(
   IN FT_HANDLE mcContext,
   IN FT_FA_RATE targetFar
);
```

Parameters

mcContext	[in] Handle to the comparison context
targetFar	[in] Target FAR. For high security, use the low value of FAR defined in **HIGH_SEC_FA_RATE**; for mid-range security, use the mid-range value of FAR defined in **MED_SEC_FA_RATE** (the default); and for low security, use the high value of FAR defined in **LOW_SEC_FA_RATE**.

Return Values

FR_OK	The function is successful
FR_ERR_NO_INIT	The fingerprint comparison module is not initialized.
FR_ERR_INVALID_PARAM	The value of the parameter **targetFar** <= 0.0 or >= 100.0, or the specified comparison context is not valid.

`FR_ERR_INVALID_CONTEXT`	The specified comparison context is not valid.
`FR_WRN_INTERNAL`	The value of the parameter `targetFar` is unacceptably high and was reduced to an internally defined value.

Library

dpHMatch.lib

MC_terminate

Terminates the fingerprint comparison module and releases the resources associated with it.

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_terminate(void);
```

Return Values

`FT_OK`	Termination was successful.

Library

dpHMatch.lib

MC_getFeaturesLen

Retrieves the size of the buffer for the fingerprint template. This function returns either the minimum or the recommended size that provides the best recognition accuracy, or both.

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_getFeaturesLen(
   IN FT_FTR_TYPE featureSetPurpose,
   IN int reserved,
   OUT int* recommendedTemplateSize,
   OUT int* minimumTemplateSize
);
```

Parameters

featureSetPurpose	[in] Feature set purpose. Specifies the purpose for which the fingerprint feature set is to be created. For a feature set to be used for enrollment, use the value **FT_PRE_REG_FTR**; for verification, use **FT_VER_FTR**; and for a fingerprint template, use **FT_REG_FTR**.
reserved	[in] This parameter is deprecated and should always be set to **0**.
recommendedTemplateSize	[out] Pointer to the memory receiving the size of the buffer for the fingerprint template recommended for best recognition accuracy, or **NULL**. If **NULL** is passed, **minimumTemplateSize** must not be **NULL**.
minimumTemplateSize	[out] Pointer to the memory receiving the minimum size of the buffer for the fingerprint template, or **NULL**. If **NULL** is passed, **recommendedTemplateSize** must not be **NULL**.

Return Values

FT_OK	Termination was successful.

Library

dpHMatch.lib

MC_generateRegFeatures

Creates a fingerprint template to be used for later comparison with a fingerprint feature set. This function, known as *fingerprint enrollment*, computes the fingerprint template using the specified number of fingerprint feature sets (**numFeatureSets**) successfully returned by the **FX_extractFeatures** function (*page 51*).

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_generateRegFeatures(
   IN FT_HANDLE mcContext,
   IN int reserved0,
   IN int numFeatureSets,
   IN int featureSetSize,
   IN FT_BYTE* featureSet[],
   IN int templateSize,
   OUT FT_BYTE* template,
   OUT FT_BYTE reserved1[],
   OUT FT_BOOL* templateCreated
);
```

Parameters

mcContext	[in] Handle to the comparison context
reserved0	[in] This parameter is deprecated and should always be set to **0**.
numFeatureSets	[in] Number of input fingerprint feature sets, which is the number specified in the **numFeatureSets** field of the structure of type **MC_SETTINGS**.
featureSetSize	[in] Size of the buffer for the fingerprint feature set (assuming that the size of each fingerprint feature set is the same)
featureSet[]	[in] Array of pointers to the locations of the buffers for each fingerprint feature set
templateSize	[in] Size of the fingerprint template
template	[out] Pointer to the location of the buffer receiving the fingerprint template
reserved1[]	[out] This parameter is deprecated and should be set to **NULL**.
templateCreated	[out] Pointer to the memory that will receive the value of whether the template is created. If the value of this parameter is **FT_TRUE**, the fingerprint template was written to **template**. If the value is **FT_FALSE**, a template was not created.

Return Values

FR_OK	The function succeeded.
FR_ERR_NO_INIT	The fingerprint comparison module is not initialized.
FR_ERR_NO_MEMORY	There is not enough memory to perform the function.
FR_ERR_BAD_INI_SETTING	Initialization settings are corrupted.
FR_ERR_INVALID_BUFFER	A buffer is not valid.
FR_ERR_INVALID_PARAM	One or more parameters are not valid.
FR_ERR_INTERNAL	An internal error occurred.

Library

dpHMatch.lib

MC_verifyFeaturesEx

Performs a one-to-one *comparison* of a fingerprint feature set with a fingerprint template produced at enrollment and makes a decision of match or non-match. This function is known as *fingerprint verification*. The function succeeds if the *comparison score* is high enough given the security level of the specified comparison context.

Syntax

```
MC_DLL_INTERFACE FT_RETCODE MC_verifyFeaturesEx(
   IN FT_HANDLE mcContext,
   IN int templateSize,
   IN OUT FT_BYTE* template,
   IN int featureSetSize,
   IN FT_BYTE* featureSet,
   IN int reserved0,
   OUT void* reserved1,
   OUT int reserved2[],
   OUT FT_VER_SCORE_PT reserved3,
   OUT double* achievedFar,
   OUT FT_BOOL* comparisonDecision
);
```

Parameters

mcContext	[in] Handle to the comparison context
templateSize	[in] Size of the fingerprint template
template	[in, out] Pointer to the location of the buffer containing the fingerprint template
featureSetSize	[in] Size of the fingerprint feature set
featureSet	[in] Pointer to the location of the buffer containing the fingerprint feature set
reserved0	[in] This parameter is deprecated and should always be **0**.
reserved1	[in] This parameter is deprecated and should always be **NULL**.
reserved2[]	[out] This parameter is deprecated and should always be **NULL**.
reserved3	[out] This parameter is deprecated and should always be **NULL**.
achievedFar	[out] Pointer to the value of the achieved FAR for this comparison. If the achieved FAR is not required, a **NULL** pointer can be passed.
comparisonDecision	[out] Pointer to the memory that will receive the comparison decision. This parameter indicates whether the comparison of the fingerprint feature set and the fingerprint template resulted in a decision of match (**FT_TRUE**) or non-match (**FT_FALSE**) at the security level of the specified comparison context.

Return Values

FR_OK	The function succeeded.
FR_ERR_NO_INIT	The fingerprint comparison module is not initialized.
FR_ERR_NO_MEMORY	There is not enough memory to perform the function.
FR_ERR_BAD_INI_SETTING	Initialization settings are corrupted.
FR_ERR_INVALID_BUFFER	An internal error occurred.
FR_ERR_INVALID_PARAM	One or more parameters are not valid.

Library

dpHMatch.lib

Data Structures

This section defines the One Touch for Windows: C/C++ Edition Core API data structures.

DP_DEVICE_INFO

Device information structure.

Syntax

```
typedef struct DP_DEVICE_INFO
{
   GUID DeviceUid;
   DP_DEVICE_UID_TYPE eUidType;
   DP_DEVICE_MODALITY eDeviceModality;
   DP_DEVICE_TECHNOLOGY eDeviceTech;
   DP_HW_INFO HwInfo;
} DP_DEVICE_INFO, * PDP_DEVICE_INFO;
```

Data Fields

`DeviceUid`	Device unique identifier
`eUidType`	Defines whether the UID is persistent or volatile
`eDeviceModality`	Defines which modality the device is being used in
`eDeviceTech`	Defines the type of technology used in the device
`HwInfo`	Describes the device hardware

DP_DEVICE_VERSION

Device hardware/firmware version number structure.

Syntax

```
typedef struct DP_DEVICE_VERSION
{
   ULONG uMajor;
   ULONG uMinor;
   ULONG uBuild;
} DP_DEVICE_VERSION;
```

Data Fields

wMajor	Major version of the product
wMinor	Minor version of the product
wRevision	Revision number of the product
uBuild	Build number of the product

DP_HW_INFO

Device hardware information structure.

Syntax

```
typedef struct DP_HW_INF
{
  unsigned int uLanguageId;
  wchar_t szVendor[DP_MAX_USB_STRING_SIZE];
  wchar_t szProduct[DP_MAX_USB_STRING_SIZE];
  wchar_t szSerialNb[DP_MAX_USB_STRING_SIZE];
  DP_DEVICE_VERSION HardwareRevision
  DP_DEVICE_VERSION FirmwareRevision;
} DP_HW_INFO, * PDP_HW_INFO;
```

Data Fields

uLanguageId	Device language
szVendor	Manufacturer name, for example, "DigitalPersona, Inc."
szProduct	Build number of device hardware/firmware

DP_PRODUCT_VERSION

DigitalPersona product version structure.

Syntax

```
typedef struct DP_PRODUCT_VERSION
{
  WORD wMajor;
  WORD wMinor;
  WORD wRevision;
  WORD wBuild;
} DP_PRODUCT_VERSION, * PDP_PRODUCT_VERSION;
```

Data Fields

wMajor	Major version of the product
wMinor	Minor version of the product
wRevision	Revision number of the product
uBuild	Build number of the product

FT_VERSION_INFO

Fingerprint feature extraction or fingerprint comparison module version information structure.

Syntax

```
typedef struct
{
  unsigned  major;
  unsigned  minor;
  unsigned  revision;
  unsigned  build;
} FT_VERSION_INFO, * FT_VERSION_INFO_PT;
```

Data Fields

Major	Major version number
Minor	Minor version number
Revision	Revision number number
Build	Build number

MC_SETTINGS

Fingerprint comparison module settings structure.

Syntax

```
typedef struct{
   int numPreRegFeatures;
} MC_SETTINGS, * MC_SETTINGS_PT;
```

Data Fields

numPreRegFeatures	Number of fingerprint feature sets required to generate a fingerprint template

Enumerations

This section defines the One Touch for Windows: C/C++ Edition Core API enumerations.

DP_ACQUISITION_PRIORITY

Defines the priority of a fingerprint sample capture operation performed by a fingerprint reader.

Syntax

```
typedef enum DP_ACQUISITION_PRIORITY
{
  DP_PRIORITY_HIGH= 1,
  DP_PRIORITY_NORMAL= 2,
  DP_PRIORITY_LOW = 3
} DP_ACQUISITION_PRIORITY;
```

Values

`DP_PRIORITY_HIGH`	Highest priority
`DP_PRIORITY_NORMAL`	Standard priority
`DP_PRIORITY_LOW`	Lowest priority

DP_DEVICE_MODALITY

Defines the modality that a fingerprint reader uses to capture fingerprint samples.

Syntax

```
typedef enum DP_DEVICE_MODALITY
{
  DP_UNKNOWN_DEVICE_MODALITY = 0,
  DP_SWIPE_DEVICE,
  DP_AREA_DEVICE,
  DP_DEVICE_MODALITY_NUM,
} DP_DEVICE_MODALITY;
```

Values

`DP_UNKNOWN_DEVICE_MODALITY`	Device modality is unknown
`DP_SWIPE_DEVICE`	Swipe mode device
`DP_AREA_DEVICE`	Area mode device
`DP_DEVICE_MODALITY_NUM`	Count of different modalities defined

DP_DEVICE_TECHNOLOGY

Defines the fingerprint reader technology.

Syntax

```
typedef enum DP_DEVICE_TECHNOLOGY
{
  DP_UNKNOWN_DEVICE_TECHNOLOGY = 0,
  DP_OPTICAL_DEVICE,
  DP_CAPACITIVE_DEVICE,
  DP_THERMAL_DEVICE,
  DP_PRESSURE_DEVICE,
  DP_DEVICE_TECHNOLOGY_NUM,
} DP_DEVICE_TECHNOLOGY;
```

Values

`DP_UNKNOWN_DEVICE_TECHNOLOGY`	The technology used in the device is not known.
`DP_OPTICAL_DEVICE`	The technology used in the device is optical.
`DP_CAPACITIVE_DEVICE`	The technology used in the device is capacitive.
`DP_THERMAL_DEVICE`	The technology used in the device is thermal.
`DP_PRESSURE_DEVICE`	The technology used in the device is pressure.
`DP_DEVICE_TECHNOLOGY_NUM`	Count of the different technologies defined.

DP_DEVICE_UID_TYPE

Defines the type of UUID identifying the device.

Syntax

```
typedef enum DP_DEVICE_UID_TYPE
{
  DP_PERSISTENT_DEVICE_UID = 0,
  DP_VOLATILE_DEVICE_UID,
} DP_DEVICE_UID_TYPE;
```

Values

`DP_PERSISTENT_DEVICE_UID`	Unique hardware identifier. Hardware dependent.
`DP_VOLATILE_DEVICE_UID`	Software generated identifier.

DP_SAMPLE_QUALITY

Defines the quality of the fingerprint sample.

Syntax

```
typedef enum DP_SAMPLE_QUALITY {
  DP_QUALITY_GOOD = 0,
  DP_QUALITY_NONE = 1,
  DP_QUALITY_TOOLIGHT = 2,
  DP_QUALITY_TOODARK = 3,
  DP_QUALITY_TOONOISY = 4,
  DP_QUALITY_LOWCONTR = 5,
  DP_QUALITY_FTRNOTENOUGH = 6,
  DP_QUALITY_NOCENTRAL = 7,
  DP_QUALITY_NOFINGER = 8,
  DP_QUALITY_TOOHIGH = 9,
  DP_QUALITY_TOOLOW = 10,
  DP_QUALITY_TOOLEFT = 11,
  DP_QUALITY_TOORIGHT = 12,
  DP_QUALITY_TOOSTRANGE = 13,
  DP_QUALITY_TOOFAST = 14,
  DP_QUALITY_TOOSKEWED = 15,
  DP_QUALITY_TOOSHORT = 16,
  DP_QUALITY_TOOSLOW = 17,
} DP_SAMPLE_QUALITY;
```

Values

DP_QUALITY_GOOD	The image is of good quality.
DP_QUALITY_NONE	There is no image.
DP_QUALITY_TOOLIGHT	The image is too light.
DP_QUALITY_TOODARK	The image is too dark.
DP_QUALITY_TOONOISY	The image is too noisy.
DP_QUALITY_LOWCONTR	The image contrast is too low.
DP_QUALITY_FTRNOTENOUGH	The image does not contain enough information.
DP_QUALITY_NOCENTRAL	The image is not centered.
DP_QUALITY_NOFINGER	The scanned object is not a finger.
DP_QUALITY_TOOHIGH	The finger was too high on the swipe sensor.
DP_QUALITY_TOOLOW	The finger was too low on the swipe sensor.
DP_QUALITY_TOOLEFT	The finger was too close to left border of the swipe sensor.
DP_QUALITY_TOORIGHT	The finger was too close to right border of the swipe sensor.
DP_QUALITY_TOOSTRANGE	The scan looks strange.
DP_QUALITY_TOOFAST	The finger was swiped too quickly.
DP_QUALITY_TOOSKEWED	The image is too skewed.
DP_QUALITY_TOOSHORT	The image is too short.
DP_QUALITY_TOOSLOW	The finger was swiped too slowly.

FT_IMG_QUALITY

Defines the image quality.

```
typedef enum
{
   FT_GOOD_IMG,
   FT_IMG_TOO_LIGHT,
   FT_IMG_TOO_DARK,
   FT_IMG_TOO_NOISY,
   FT_LOW_CONTRAST,
   FT_UNKNOWN_IMG_QUALITY
} FT_IMG_QUALITY, *FT_IMG_QUALITY_PT;
```

Values

`FT_GOOD_IMG`	The fingerprint image quality is good.
`FT_IMG_TOO_LIGHT`	The fingerprint image is too light.
`FT_IMG_TOO_DARK`	The fingerprint image is too dark.
`FT_IMG_TOO_NOISY`	The fingerprint image is too blurred.
`FT_LOW_CONTRAST`	The fingerprint image contrast is too low.
`FT_UNKNOWN_IMG_QUALITY`	The fingerprint image quality is undetermined.

FT_FTR_QUALITY

Defines the fingerprint features quality.

```
typedef enum
{
   FT_GOOD_FTR,
   FT_NOT_ENOUGH_FTR,
   FT_NO_CENTRAL_REGION,
   FT_UNKNOWN_FTR_QUALITY,
   FT_AREA_TOO_SMALL
} FT_FTR_QUALITY, * FT_FTR_QUALITY_PT;
```

Values

`FT_GOOD_FTR`	The fingerprint features quality is good.
`FT_NOT_ENOUGH_FTR`	There are not enough fingerprint features.
`FT_NO_CENTRAL_REGION`	The fingerprint image does not contain the central portion of the finger.
`FT_UNKNOWN_FTR_QUALITY`	The fingerprint features quality is undetermined.
`FT_AREA_TOO_SMALL`	The fingerprint image area is too small.

FT_FTR_TYPE

Defines the feature set purpose.

Syntax

```
typedef enum
{
   FT_PRE_REG_FTR,
   FT_REG_FTR,
} FT_FTR_TYPE;
```

Values

`FT_PRE_REG_FTR`	Value for a fingerprint feature set to be used for enrollment
`FT_REG_FTR`	Value for a fingerprint template
`FT_VER_FTR`	Value for a fingerprint feature set to be used for verification

Type Definitions and Constants

This section defines the One Touch for Windows: C/C++ Edition Core API type definitions and constants.

DFLT_FA_RATE MED_SEC_FA_RATE

Default value for target FAR.

Syntax

```
#define DFLT_FA_RATE MED_SEC_FA_RATE
```

DP_SAMPLE_TYPE_IMAGE

Type of fingerprint sample needed. This value is used in the **uSampleType** parameter of the **DPFPCreateAcquisition** function (*page 38*).

Syntax

```
#define DP_SAMPLE_TYPE_IMAGE 4
```

FT_FA_RATE

Target false accept rate (FAR). These are percentages, that is, a value of 0.1 means 0.1% = 1/1000. */

Syntax

```
typedef double FT_FA_RATE;
```

HDPOPERATION

Operation handle.

Syntax

```
typedef unsigned long HDPOPERATION
```

HIGH_SEC_FA_RATE

High security/low value for target FAR.

Syntax

```
#define HIGH_SEC_FA_RATE 0.0001f
```

LOW_SEC_FA_RATE

Low security/high value for target FAR.

Syntax

```
#define LOW_SEC_FA_RATE 0.0100f
```

MED_SEC_FA_RATE

Mid-range security/mid-range value for target FAR.

Syntax

```
#define MED_SEC_FA_RATE 0.0010f
```

User Interface API Reference 6

This chapter provides a reference to the User Interface API (DPUIAPI) wrapper that simplifies access to the entire functionality available in the Core API described in the previous chapter. The wrapper provides a premade user interface that handles device component, fingerprint enrollment, and fingerprint verification tasks through a few simple functions and two callbacks.

Functions

DPEnrollUI

This function displays the user interface for enrolling the fingerprints and returns after closing of the user interface. It does not store the fingerprint template; instead, it calls the application-defined function **DPENROLLMENTPROC** for each enrollment or deletion of a fingerprint.

Syntax

```
DPFPUI_STDAPI DPEnrollUI(HWND hParentWnd,
   USHORT usMaxEnrollFingerCount,
   PULONGpulEnrolledFingersMask,
   DPENROLLMENTPROC dpEnrollmentProc,
   LPVOID pUserData
);
```

Parameters

hParentWnd	[in] Handle to the parent window.
usMaxEnrollFingerCount	[in] Maximum number of fingers allowed to be enrolled. The value should be between 1 and 10 (both inclusive).
pulEnrolledFingersMask	[in, out] Bitwise mask that specifies the fingers enrolled. For possible values, see *Table 4*.
dpEnrollmentProc	[in] Pointer to the function to be notified when a fingerprint template is available for enrollment.
pUserData	[in] The pointer to the user data.

The **pulEnrolledFingersMask** parameter contains a combination of the values representing a user's enrolled fingerprints. For example, if a user's right index fingerprint and right middle fingerprint are enrolled, the value of this property is 00000000 011000000, or 192.

Table 4. Values for the **pulEnrolledFingersMask** parameter

Finger	Binary Representation	Integer Representation
Left little finger	000000000 000000001	1
Left ring finger	000000000 000000010	2
Left middle finger	000000000 000000100	4
Left index finger	000000000 000001000	8
Left thumb	000000000 000010000	16
Right thumb	000000000 000100000	32
Right index finger	000000000 001000000	64
Right middle finger	000000000 010000000	128
Right ring finger	000000000 100000000	256
Right little finger	000000001 000000000	512

Return Values

S_OK	Function successfully completed.

Library

DPFPUI.dll

DPVerifyUI

Displays the fingerprint verification user interface. The title, text, and banner image of the fingerprint verification user interface can be customized.

Syntax

```
DPFPUI_STDAPI DPVerifyUI(
   HWND hParentWnd,
   DPVERIFYPROC dpVerifyProc,
   LPCWSTR lpszCaption,
   LPCWSTR lpszText,
   HBITMAP hBanner,
   LPVOID pUserData
);
```

Parameters

`hParentWnd`	[in] Handle to the parent window.
`dpVerifyProc`	[in] Pointer to the callback function.
`lpszCaption`	[in] The caption of the dialog box.
`lpszText`	[in] The text of the dialog box.
`hBanner`	[in] The custom banner bitmap.
`pUserData`	[in] The pointer to the user data.

Return Values

`S_OK`	Fingerprint verification user interface successfully displayed.
`0x800704c7`	Fingerprint verification canceled by user.
`E_ABORT`	Fingerprint verification was aborted by callback function.

Library

DPFPUI.dll

Callbacks

A callback is executable code that is passed as an argument to other code. It allows a lower-level software layer to call a subroutine (or function) defined in a higher-level layer.

DPENROLLMENTPROC

This is the application-provided callback function. This function is called while enrolling a new fingerprint or deleting an enrolled fingerprint. The application should handle the storing of new fingerprint templates for comparison or deleting of an enrolled fingerprint template in this callback. The application can display its own success or error messages.

```
typedef HRESULT (CALLBACK *DPENROLLMENTPROC)(
   HWND hParentWnd,
   DP_ENROLLMENT_ACTION enrollmentAction,
   UINT uiFingerIndex,
   PDATA_BLOB pFingerprintTemplate,
   LPVOID pUserData
);
```

Parameters

`hParentWnd`	[in] Handle to the parent window.
`enrollmentAction`	[in] Specifies whether to enroll the fingerprint or delete it. Values are `DP_ENROLLMENT_ADD` or `DP_ENROLLMENT_DELETE`.
`uiFingerIndex`	[in] The index of the fingerprint to be enrolled, as defined in ANSI/NIST-ITL 1. For possible values, see *Table 5*.
`pFingerprintTemplate`	[in] If the `enrollmentAction` parameter is `DP_ENROLLMENT_ADD`, then this contains a pointer to the fingerprint template. Otherwise it is `NULL`.
`pUserData`	[in] Pointer to the user data.

The `uiFingerIndex` parameter contains the index value of the finger associated with a fingerprint template to be enrolled or with a fingerprint template to be deleted, as defined in ANSI/NIST-ITL 1. The index values are assigned to the graphical representation of the fingers on the hands in the user interface. All possible values are listed in *Table 5*.

Table 5. Values for the **`uiFingerIndex`** parameter

Finger	Index Value	Finger	Index Value
Right thumb	1	Left thumb	6
Right index finger	2	Left index finger	7
Right middle finger	3	Left middle finger	8
Right ring finger	4	Left ring finger	9
Right little finger	5	Left little finger	10

Return Values

`S_OK`	The fingerprint template was successfully saved.
`0x800704c7`	The operation could not be completed. A retry should be performed.

Library

DPFPUI.dll

DPVERIFYPROC

This is an application-provided callback function. It is called when a fingerprint feature set is ready for comparison. The application should handle the comparison of this fingerprint feature set against the fingerprint templates.

Syntax

```
typedef HRESULT (CALLBACK *DPVERIFYPROC)(
  HWND hParentWnd,
  PDATA_BLOB pVerificationFeatureSet,
  LPVOID pUserData
);
```

Parameters

`hParentWnd`	[in] Handle to the parent window.
`pVerificationFeatureSet`	[in] Pointer to the fingerprint feature set to be verified.
`pUserData`	[in] Pointer to the user data.

Return Values

S_OK	The fingerprint feature set to be verified matches one of the fingerprint templates.
0x800704c7	The fingerprint feature set to be verified did not match any of the fingerprint templates. A retry should be performed.

Library

DPFPUI.dll

Enumerations

DP_ENROLLMENT_ACTION

Defines the requested fingerprint enrollment action.

Syntax

```
typedef enum
{
    DP_ENROLLMENT_ADD,
    DP_ENROLLMENT_DELETE
} DP_ENROLLMENT_ACTION;
```

Values

`DP_ENROLLMET_ADD`	Enroll a fingerprint template.
`DP_ENROLLMET_DELETE`	Delete a fingerprint template.

Events Notifications and Return Codes 7

This chapter defines the notification events and return codes used within the One Touch for Windows: C/C++ Edition SDK.

Events Notifications

During the creation of an operation, the client application specifies the handle of the window to be notified on operation-related events as well as the window message to be sent as a notification. The **wParam** of the message specifies the event type. The value of **lParam** is event-specific.

Value	Defines	Description
0	**WN_COMPLETED**	Operation completed successfully. The fingerprint image is returned in **lParam** as pointer to **DATA_BLOB** structure.
1	**WN_ERROR**	An error occurred. The error code is returned in **lParam**.
2	**WN_DISCONNECT**	The device has been disconnected. The pointer to device UID is returned in **lParam**.
3	**WN_RECONNECT**	The device has been reconnected. The pointer to device UID is returned in **lParam**.
4	**WN_SAMPLEQUALITY**	Provides information about the quality of the fingerprint image. **lParam** contains the fingerprint image quality listed in the enum of type **DP_SAMPLE_QUALITY**.
5	**WN_FINGER_TOUCHED**	The device has been touched.
6	**WN_FINGER_GONE**	The finger has been removed from the device.
7	**WN_IMAGE_READY**	An image is ready from the device. The pointer to device UID is returned in **lParam**.
10	**WN_OPERATION_STOPPED**	Sent when an operation was stopped by calling **DPFPStopAcquisition**.

Return Codes

Value	Return Code	Description
0	FT_OK	The function succeeded.
1	FT_WRN_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module are not initialized.
8	FT_WRN_INTERNAL	An internal error occurred.
9	FT_WRN_KEY_NOT_FOUND	The fingerprint feature extraction module or the fingerprint comparison module could not find an initialization setting.
11	FT_WRN_UNKNOWN_DEVICE	The fingerprint reader is not known.
12	FT_WRN_TIMEOUT	The function has timed out.
-1	FT_ERR_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module is not initialized.
-2	FT_ERR_INVALID_PARAM	One or more parameters are not valid.
-3	FT_ERR_NOT_IMPLEMENTED	The called function was not implemented
-4	FT_ERR_IO	A generic I/O file error occurred.
-7	FT_ERR_NO_MEMORY	There is not enough memory to perform the action.
-8	FT_ERR_INTERNAL	An unknown internal error occurred.
-9	FT_ERR_BAD_INI_SETTING	Initialization settings are corrupted.
-10	FT_ERR_UNKNOWN_DEVICE	The fingerprint reader is not known.
-11	FT_ERR_INVALID_BUFFER	A buffer is not valid.
-16	FT_ERR_FEAT_LEN_TOO_SHORT	The specified fingerprint feature set or fingerprint template buffer size is too small.
-17	FT_ERR_INVALID_CONTEXT	The given context is not valid.
-29	FT_ERR_INVALID_FTRS_TYPE	The feature set purpose is not valid.
-32	FT_ERR_FTRS_INVALID	Decrypted fingerprint features are not valid. Decryption may have failed.
-33	FT_ERR_UNKNOWN_EXCEPTION	An unknown exception occurred.

Developing Citrix-aware applications   8

This SDK includes support for fingerprint authentication through Windows Terminal Services (including Remote Desktop Connection) and through a Citrix connection to Metaframe Presentation Server using a client from the Citrix Presentation Server Client package.

The following types of Citrix clients are supported for fingerprint authentication:

- Program Neighborhood
- Program Neighborhood Agent
- Web Client

In order to utilize this support, your application (or the end-user) will need to copy a file to the client computer and register it. The name of the file is DPICACnt.dl, and it is located in the "Misc\Citrix Support" folder in the product package.

To deploy the DigitalPersona library for Citrix support:

1. Locate the DPICACnt.dll file in the "Misc\Citrix Support" folder of your software package..

2. Copy the file to the folder on the client computer where the Citrix client components are located (i.e. for the Program Neighborhood client it might be the "Program Files\Citrix\ICA Client" folder).

3. Using the regsvr32.exe program, register the DPICACnt.dll library.

If you have several Citrix clients installed on a computer, deploy the DPICACnt.dll library to the Citrix client folder for each client.

If your application will also be working with Pro Workstation 4.2.0 and later or Pro Kiosk 4.2.0 and later, you will need to inform the end-user's administrator that they will need to enable two Group Policy Objects (GPOs), "Use DigitalPersona Pro Server for authentication" and "Allow Fingerprint Data Redirection". For information on how to enable these policies, see the "DigitalPersona Pro for AD Guide.pdf" located in the DigitalPersona Pro Server software package.

Redistribution 9

You may redistribute the files in the RTE\Install and the Redist folders in any of the One Touch for Windows SDK software packages to your end users pursuant to the terms of the end user license agreement (EULA), attendant to the software and located in the Docs folder in the SDK software package.

When you develop a product based on the One Touch for Windows SDK, you need to provide the redistributables to your end users. These files are designed and licensed for use with your application. You may include the installation files located in the RTE\Install folder in your application, or you may incorporate the redistributables directly into your installer. You may also use the merge modules located in the Redist folder in the SDK software package to create your own MSI installer.

Per the terms of the EULA, DigitalPersona grants you a non-transferable, non-exclusive, worldwide license to redistribute, either directly or via the respective merge modules, the following files contained in the RTE\Install and Redist folders in the One Touch for Windows SDK software package to your end users and to incorporate these files into derivative works for sale and distribution:

RTE\Install Folder

- InstallOnly.bat
- Setup.exe
- Setup.msi
- UninstallOnly.bat

Redist Folder

- DpCore.msm

  This merge module contains the following files:

  - Dpcoper2.dll
  - Dpdevice2.dll
  - Dpfpapi.dll
  - Dphostw.exe
  - Dpmux.dll
  - Dpmsg.dll
  - Dpclback.dll
  - DPCrStor.dll

- DpCore_x64.msm

  This merge module contains the following files:

  - Dpcoper2.dll

  - Dpdevice2.dll

  - Dpfpapi.dll

  - Dphostw.exe

  - Dpmux.dll

  - Dpclback.dll

  - DPCrStor.dll

  - x64\Dpmsg.dll

- DpDrivers.msm

  This merge module contains the following files:

  - Dpd00701x64.dll

  - Dpdevctlx64.dll

  - Dpdevdatx64.dll

  - Dpersona_x64.cat

  - Dpersona_x64.inf

  - Dpi00701x64.dll

  - Dpinst32.exe

  - Dpinst64.exe

  - Usbdpfp.sys

  - Dpersona.cat

  - Dpersona.inf

  - Dpdevctl.dll

  - Dpdevdat.dll

  - Dpk00701.sys

  - Dpk00303.sys

  - Dpd00303.dll

  - Dpd00701.dll

  - Dpi00701.dll

- DpFpRec.msm

  This merge module contains the following files:

  - Dphftrex.dll

  - Dphmatch.dll

- DpFpRec_x64.msm

  This merge module contains the following files:

  - *<system folder>*\Dphftrex.dll

  - *<system folder>*\Dphmatch.dll

  - *<system64 folder>*\Dphftrex.dll

  - *<system64 folder>*\Dphmatch.dll

- DPFpUI.msm

  This merge module contains the following file:

  - Dpfpui.dll

- DPFpUI_x64.msm

  This merge module contains the following file:

  - *<system folder>*\Dpfpui.dll

  - *<system64 folder>*\Dpfpui.dll

- DpProCore.msm

  This merge module contains the following files:

  - Dpdevts.dll

  - Dpsvinfo2.dll

  - Dptsclnt.dll

- •DpOTCOMActX.msm

  This merge module contains the following files:

  - DPFPShrX.dll

  - DPFPDevX.dll

  - DPFPEngX.dll

  - DPFPCtlX.dll

- •DpOTCOMActX_x64.msm

  This merge module contains the following files:

- DPFPShrX.dll

- DPFPDevX.dll

- DPFPEngX.dll

- DPFPCtlX.dll

- x64\DpFpCtlX.dll

- x64\DpFpDevX.dll

- x64\DpFpEngX.dll

- x64\DpFpShrX.dll

- •DpOTDotNET.msm

  This merge module contains the following files:

  - DPFPShrNET.dll

  - DPFPDevNET.dll

  - DPFPEngNET.dll

  - DPFPVerNET.dll

  - DPFPGuiNET.dll

  - DPFPCtlXLib.dll

  - DPFPCtlXTypeLibNET.dll

  - DPFPCtlXWrapperNET.dll

  - DPFPShrXTypeLibNET.dll

Fingerprint Reader Documentation

You may redistribute the documentation included in the Redist folder of any One Touch for Windows SDK software package to your end users pursuant to the terms of this section and of the EULA, attendant to the software and located in the Docs folder in the SDK software package.

Hardware Warnings and Regulatory Information

If you distribute DigitalPersona U.are.U fingerprint readers to your end users, you are responsible for advising them of the warnings and regulatory information included in the Warnings and Regulatory Information.pdf file in the Redist folder of any One Touch for Windows SDK software package. You may copy and redistribute the language, including the copyright and trademark notices, set forth in the Warnings and Regulatory Information.pdf file.

Fingerprint Reader Use and Maintenance Guide

The DigitalPersona U.are.U fingerprint reader use and maintenance guides, DigitalPersona Reader Maintenance Touch.pdf and DigitalPersona Reader Maintenance Swipe.pdf, are located in the Redist folder in the One Touch for Windows SDK software package. You may copy and redistribute the DigitalPersona Reader Maintenance Touch.pdf and the DigitalPersona Reader Maintenance Swipe.pdf files, including the copyright and trademark notices, to those who purchase a U.are.U module or fingerprint reader from you.

Setting the False Accept Rate $\quad$ *A*

This appendix is for developers who want to specify a false accept rate (FAR) other than the default used by the DigitalPersona Fingerprint Recognition Engine.

False Accept Rate (FAR)

The false accept rate (FAR), also known as the security level, is the proportion of fingerprint verification operations by authorized users that incorrectly returns a comparison decision of match. The FAR is typically stated as the ratio of the expected number of false accept errors divided by the total number of verification attempts, or the probability that a biometric system will falsely accept an unauthorized user. For example, a probability of 0.001 (or 0.1%) means that out of 1,000 verification operations by authorized users, a system is expected to return 1 incorrect match decision. Increasing the probability to, say, 0.0001 (or 0.01%) changes this ratio from 1 in 1,000 to 1 in 10,000.

Increasing or decreasing the FAR has the opposite effect on the false reject rate (FRR), that is, decreasing the rate of false accepts increases the rate of false rejects and vice versa. Therefore, a high security level may be appropriate for an access system to a secured area, but may not be acceptable for a system where convenience or easy access is more significant than security.

Representation of Probability

Probability should always be in the range from 0 to 1. Some common representations of probability are listed in column one of *Table 2*. The value in the third row represents the current default value used by the DigitalPersona Fingerprint Recognition Engine, which offers a mid-range security level. The value in the second row represents a typical high FAR/low security level, and the value in the fourth row represents a typical low FAR/high security level.

Table 2. Common values of probability

Decimal (0 to 1)	Percent (0 to 100%)	Fraction (1/n[1] to 1)
0.001	0.1%	1/1000
0.0001	0.01%	1/10000
0.00001	0.001%	1/100000
0.000001	0.0001%	1/1000000

1. Where n is equal to infinity.

Requested FAR

You specify the value of the FAR using the `targetFar` parameter of the `MC_setSecurityLevel` function. While you can request any value from 0 to 100%, it is not guaranteed that the Engine will fulfill the request exactly. The Engine implementation makes the best effort to accommodate the request by internally setting the value closest to that requested within the restrictions it imposes for security.

The following sample code sets the FAR to a value of 0.005%.

```
...

//Sets the FAR to 0.005%
rc = MC_setSecurityLevel(mcContext, 0.005);
```

Achieved FAR

The actual value of the FAR achieved for a particular verification operation is returned in the `achievedFar` parameter of `MC_verifyFeaturesEx` function as a probability value between 0 and 1. This value is typically much smaller than the requested FAR due to the accuracy of the DigitalPersona Fingerprint Recognition Engine. The requested FAR specifies the maximum value of the FAR to be used by the Engine in making the verification decision. The actual FAR achieved by the Engine when conducting a legitimate comparison is usually a much lower value. The Engine implementation may choose the range and granularity for the achieved FAR. If you make use of this value in your application, for example, by combining it with other achieved FARs, you should use it with caution, as the granularity and range may change between versions of DigitalPersona SDKs without notice.

Testing

Although you may achieve the desired values of the FAR in your development environment, it is not guaranteed that your application will achieve the required security level in real-world situations. Even though the Engine is designed to make its best effort to accurately implement the probability estimates, it is recommended that you conduct system-level testing to determine the actual operating point and accuracy in a given scenario. This is even more important in systems where multiple biometric factors are used for identification.

Platinum SDK Enrollment Template Conversion   *B*

This appendix is for Platinum SDK users who need to convert their Platinum SDK registration templates to a format that is compatible with the SDKs that are listed in *Fingerprint Template Compatibility* on *page 5*.

Sample code is included below for C++ and Visual Basic.

Platinum SDK Enrollment Template Conversion for Microsoft Visual C++

Use *Code Sample 1* in applications developed in Microsoft Visual C++ to convert DigitalPersona Platinum SDK registration templates.

Code Sample 1. Platinum SDK Template Conversion for Microsoft Visual C++

```cpp
#import "DpSdkEng.tlb" no_namespace, named_guids, raw_interfaces_only
#include <atlbase.h>

bool PlatinumTOGold(unsigned char* platinumBlob, int platinumBlobSize,
                    unsigned char* goldBlob, int goldBufferSize,
                    int* goldTemplateSize)
{
    // Load the byte array into FPTemplate Object
    // to create Platinum template object
    SAFEARRAYBOUND rgsabound;
    rgsabound.lLbound = 0;
    rgsabound.cElements = platinumBlobSize;

    CComVariant varVal;
    varVal.vt = VT_ARRAY | VT_UI1;
    varVal.parray = SafeArrayCreate(VT_UI1, 1, &rgsabound);

    unsigned char* data;
    if (FAILED(SafeArrayAccessData(varVal.parray, (void**)&data)))
        return false;

    memcpy(data, platinumBlob, platinumBlobSize);
    SafeArrayUnaccessData(varVal.parray);

    IFPTemplatePtr pIFPTemplate(__uuidof(FPTemplate));

    if (pIFPTemplate == NULL)
        return false;
```

Code Sample 1. Platinum SDK Template Conversion for Microsoft Visual C++ *(continued)*

```
    AIErrors error;
    if (FAILED(pIFPTemplate->Import(varVal, &error)))
        return false;

    if (error != Er_OK)
    return false;

    // Now pIFPTemplate contains the Platinum template.
    // Use TemplData property to get the Gold Template out.
    CComVariant varValGold;

    if (FAILED(pIFPTemplate->get_TemplData(&varValGold)))
        return false;

    unsigned char* dataGold;
    if (FAILED(SafeArrayAccessData(varValGold.parray, (void**)&dataGold)))
        return false;

    int blobSizeRequired = varValGold.parray->rgsabound->cElements *
                          varValGold.parray->cbElements;
    *goldTemplateSize = blobSizeRequired;

    if (goldBufferSize < blobSizeRequired) {
        SafeArrayUnaccessData(varValGold.parray);
        return false;
    }

    memcpy(goldBlob, dataGold, blobSizeRequired);

    SafeArrayUnaccessData(varValGold.parray);

    return true;

}
```

Platinum SDK Enrollment Template Conversion for Visual Basic 6.0

Use *Code Sample 2* in applications developed in Microsoft Visual Basic 6.0 to convert DigitalPersona Platinum SDK enrollment templates.

Code Sample 2. Platinum SDK Template Conversion for Visual Basic 6.0

```
Public Function PlatinumToGold(platinumTemplate As Variant) As Byte()
Dim pTemplate As New FPTemplate
Dim vGold As Variant
Dim bGold() As Byte

Dim er As DpSdkEngLib.AIErrors
er = pTemplate.Import(platinumTemplate)
If er <> Er_OK Then PlatinumToGold = "": Exit Function
vGold = pTemplate.TemplData
bGold = vGold
PlatinumToGold = bGold
End Function
```

Get/Set Device Parameters *C*

This appendix is for developers who want to specify and retrieve a device-specific private key on any of DigitalPersona's family of fingerprint readers (4000B and later).

Overview

All of DigitalPersona's current family of fingerprint readers, beginning with the U.are.U 4000B series, have a dedicated memory location that can be used to set a private key on the device and then retrieve the key programatically. This feature can be used to "lock in" the use of a specific device with your applications. This is most often used to ensure that a specific feature set and/or tested hardware device is the one that your application expects.

Parameters

There are two Device Parameter functions accessible through the DPFPAPI library (DPFPAPI.dll); DPFPSetDeviceParameter() and DPFPGetDeviceParameter(). The prototypes for these functions are available in `dpfpapi.h`, and are shown below for your convenience.

```
DPFP_STDAPI DPFPSetDeviceParameter(
    REFGUID DevUID,
    unsigned long ulParamID,
    const DATA_BLOB* pData
);

DPFP_STDAPI DPFPGetDeviceParameter(
    REFGUID DevUID,
    unsigned long ulParamID,
    DATA_BLOB* pData
);
```

These two functions set and get (read) the value stored on the fingerprint reader. They both require the use of a device GUID, which can be retrieved through **DPFPEnumerateDevices** (see page *41*).

A parameter ID is also needed (`ulParamID`). The currently supported parameter IDs are:

`FT_SET_CLIENT_PRIVATE_KEY` - Writes a private key to a device. The device then needs to be recycled prior to reading the value.

`FT_GET_CLIENT_PRIVATE_KEY` - Reads the private key from a device.

Glossary

biometric system

An automatic method of identifying a person based on the person's unique physical and/or behavioral traits, such as a fingerprint or an iris pattern, or a handwritten signature or a voice.

comparison

The estimation, calculation, or measurement of similarity or dissimilarity between fingerprint feature set(s) and fingerprint template(s).

comparison score

The numerical value resulting from a comparison of fingerprint feature set(s) with fingerprint template(s). Comparison scores can be of two types: similarity scores or dissimilarity scores.

context

A temporary object used for passing data between the steps of multi-step programming operations.

DigitalPersona Fingerprint Recognition Engine

A set of mathematical algorithms formalized to determine whether a fingerprint feature set matches a fingerprint template according to a specified security level in terms of the false accept rate (FAR).

enrollee

See **fingerprint data subject**.

enrollment

See **fingerprint enrollment**.

false accept rate (FAR)

The proportion of fingerprint verification transactions by fingerprint data subjects not enrolled in the system where an incorrect decision of match is returned.

false reject rate (FRR)

The proportion of fingerprint verification transactions by fingerprint enrollment subjects

against their own fingerprint template(s) where an incorrect decision of non-match is returned.

features

See **fingerprint features**.

fingerprint

An impression of the ridges on the skin of a finger.

fingerprint capture device

A device that collects a signal of a fingerprint data subject's fingerprint characteristics and converts it to a fingerprint sample. A device can be any piece of hardware (and supporting software and firmware). In some systems, converting a signal from fingerprint characteristics to a fingerprint sample may include multiple components such as a camera, photographic paper, printer, digital scanner, or ink and paper.

fingerprint characteristic

Biological finger surface details that can be detected and from which distinguishing and repeatable fingerprint feature set(s) can be extracted for the purpose of fingerprint verification or fingerprint enrollment.

fingerprint data

Either the fingerprint feature set, the fingerprint template, or the fingerprint sample.

fingerprint data storage subsystem

A storage medium where fingerprint templates are stored for reference. Each fingerprint template is associated with a fingerprint enrollment subject. Fingerprint templates can be stored within a fingerprint capture device; on a portable medium such as a smart card; locally, such as on a personal computer or a local server; or in a central database.

fingerprint data subject

A person whose fingerprint sample(s), fingerprint feature set(s), or fingerprint template(s) are present within the fingerprint recognition system at any time.

Fingerprint data can be either from a person being recognized or from a fingerprint enrollment subject.

fingerprint enrollment

a. In a fingerprint recognition system, the initial process of collecting fingerprint data from a person by extracting the fingerprint features from the person's fingerprint image for the purpose of enrollment and then storing the resulting data in a template for later comparison.

b. The system function that computes a fingerprint template from a fingerprint feature set(s).

fingerprint enrollment subject

The fingerprint data subject whose fingerprint template(s) are held in the fingerprint data storage subsystem.

fingerprint feature extraction

The system function that is applied to a fingerprint sample to compute repeatable and distinctive information to be used for fingerprint verification or fingerprint enrollment. The output of the fingerprint feature extraction function is a fingerprint feature set.

fingerprint features

The distinctive and persistent characteristics from the ridges on the skin of a finger. *See also* **fingerprint characteristics**.

fingerprint feature set

The output of a completed fingerprint feature extraction process applied to a fingerprint sample. A fingerprint feature set(s) can be produced for the purpose of fingerprint verification or for the purpose of fingerprint enrollment.

fingerprint image

A digital representation of fingerprint features prior to extraction that are obtained from a fingerprint reader. *See also* **fingerprint sample**.

fingerprint reader

A device that collects data from a person's fingerprint features and converts it to a fingerprint image.

fingerprint recognition system

A biometric system that uses the distinctive and persistent characteristics from the ridges of a finger, also referred to as *fingerprint features*, to distinguish one finger (or person) from another.

fingerprint sample

The analog or digital representation of fingerprint characteristics prior to fingerprint feature extraction that are obtained from a fingerprint capture device. A fingerprint sample may be raw (as captured), or intermediate (after some processing).

fingerprint template

The output of a completed fingerprint enrollment process that is stored in a fingerprint data storage subsystem. Fingerprint templates are stored for later comparison with a fingerprint feature set(s).

fingerprint verification

a. In a fingerprint recognition system, the process of extracting the fingerprint features from a person's fingerprint image provided for the purpose of verification, comparing the resulting data to the template generated during enrollment, and deciding if the two match.

b. The system function that performs a one-to-one comparison and makes a decision of match or non-match.

match

The decision that the fingerprint feature set(s) and the fingerprint template(s) being compared are from the same fingerprint data subject.

non-match

The decision that the fingerprint feature set(s) and the fingerprint template(s) being compared are not from the same fingerprint data subject.

one-to-one comparison

The process in which recognition fingerprint feature set(s) from one or more fingers of one fingerprint data subject are compared with fingerprint template(s) from one or more fingers of one fingerprint data subject.

repository

See **fingerprint data storage subsystem**.

security level

The target false accept rate for a comparison context. *See also* **FAR**.

verification

See **fingerprint verification**.

Index

setting
    function for *58*
    in typical verification workflow *30*
targetFar parameter
    defined *57*
    important notice to read Appendix A before setting *58*
template compatibility
    See fingerprint template compatibility
terminating
    comparison module
        function for *59*
        in typical enrollment workflow *29*
        in typical verification workflow *34*
    feature extraction module
        in typical enrollment workflow *29*
        in typical verification workflow *34*
typefaces, uses of
    bold *3*
    Courier bold *3*
    italics *3*
typographical conventions *3*

U

uninstalling RTE silently *16*
updates for DigitalPersona software products, URL for
        downloading *4*
URLs
    DigitalPersona Developer Connection Forum *4*
    Updates for DigitalPersona Software Products *4*
use and maintenance guides for fingerprint readers,
        redistributing *90*
Use DigitalPersona Pro Server for authentication *85*
using
    device component API functions *23*
    fingerprint recognition component API functions *24—*
        34

V

verification
    See fingerprint verification

W

Windows Terminal Services *1*