DigitalPersona, Inc.

One Touch® for Windows® SDK

.NET Edition

Version 1.6

Developer Guide

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: June 22, 2010 (1.6.0)

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.

Also note that the DigitalPersona One Touch I.D. SDK includes the One Touch for Windows RTE, .NET documentation and .NET samples as well; and can be used to implement a full-fledged biometrics product encompassing fingerprint collection, enrollment, and verification. We strongly suggest that OTID developers use this embedded version of OTW.

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

To take advantage of this feature, your fingerprint-enabled application must run on the Terminal Services or Citrix server, not on the client. See also, *Developing Citrix-aware applications* beginning on page *104*.

Target Audience

This guide is for developers who have a working knowledge of the C# or Microsoft® Visual Basic® 2005 programming language, and the Microsoft® Visual Studio® 2005 (or later) development environment.

Chapter Overview

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

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

Chapter 3, *Installation*, contains instructions for installing the various components of the product and identifies the files and folders that are installed on your hard disk.

Chapter 4, *Overview*, introduces One Touch for Windows SDK: .NET Edition terminology and concepts. This chapter also includes typical workflow diagrams and explanations of the One Touch for Windows: .NET Edition API components used to perform the tasks in the workflows.

Chapter 5, *API Reference*, defines the components that are used for developing applications based on the One Touch for Windows: .NET Edition API.

Chapter 6, *Graphical User Interfaces*, describes the functionality of the graphical user interfaces included with the DPFP.Gui.Enrollment.EnrollmentControl and DPFP.Gui.Verification.VerificationControl objects.

Chapter 8, *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: .NET 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: .NET Edition.

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

Document Conventions

This section defines the notational, typographical, and naming 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
Bold	Used for keystrokes and window and dialog box elements and to indicate data types	Click **Fingerprint Enrollment**. The **Fingerprint Enrollment** dialog box appears. **String** that contains a fingerprint reader serial number
`Courier bold`	Used to indicate computer programming code	Check the `TemplateStatus` property after each call to the `AddFeatures` method. Initialize a new instance of the `DPFP.Capture.Capture` class.
Italics	Used for emphasis or to introduce new terms If you are viewing this document online, clicking text in italics may also activate a hypertext link to other areas in this guide or to URLs.	This section includes illustrations of *typical* fingerprint enrollment and fingerprint verification workflows. (emphasis) A *fingerprint* is an impression of the ridges on the skin of a finger. (new term) See *Installing the SDK* on *page 8*. (link to heading and page)

Naming Conventions

DPFP stands for *DigitalPersona Fingerprint*.

Additional Resources

You can refer to the resources in this section to assist you in using the One Touch for Windows SDK: .NET 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 One Touch for Window 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: .NET 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
- Microsoft .NET Framework (version 2 and later), which is required for .NET projects to run and is obtainable from Microsoft
- USB connector on the computer where the fingerprint reader is to be connected

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.

Supported DigitalPersona Hardware Products

The One Touch for Windows SDK: .NET 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

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 114* 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: .NET Edition using one of the sample applications provided as part of the One Touch for Windows SDK.

The application is a Microsoft Visual Studio 2005 project that demonstrates the functionality of the graphical user interfaces included in the `DPFP.Gui.Enrollment.EnrollmentControl` and `DPFP.Gui.Verification.VerificationControl` objects. The graphical user interfaces are described in more detail in *DPFP.Gui.Enrollment Graphical User Interface* on *page 94* and *DPFP.Gui.Verification Graphical User Interface* on *page 103*.

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 18*.

Install the Software

Before you can use the sample application, you must install the One Touch for Windows SDK: .NET Edition, which includes the runtime environment (RTE).

To install the One Touch for Windows SDK: .NET Edition

1. In the SDK folder in the software package, open the Setup.exe file, and then click **Next**.
2. Follow the installation instructions as they appear.
3. Restart your computer.

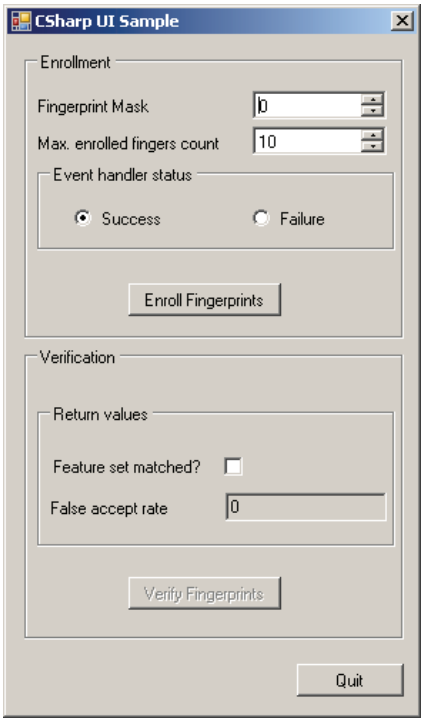
Connect the Fingerprint Reader

Connect the fingerprint reader into 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

Open the UISupportSample CS.exe file located in the *<destination folder>*\One Touch SDK\.NET\Samples\Visual Studio 2005\CSharp\UI Support\Release folder.

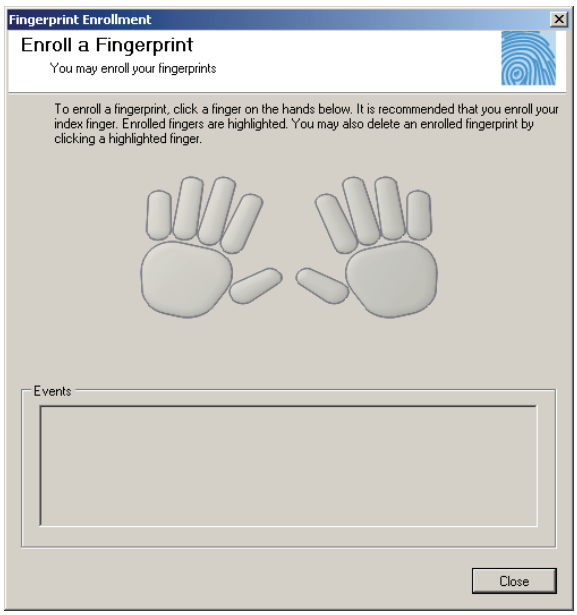
The **CSharp UI Sample** dialog box appears.

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

To enroll a fingerprint

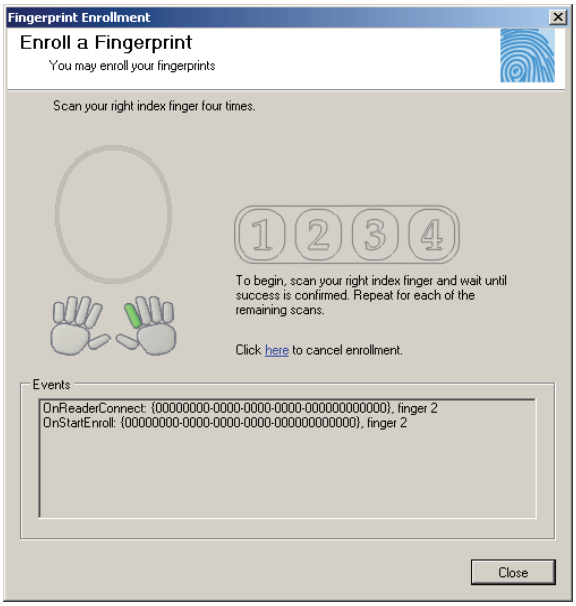
1. In the **CSharp UI Sample** dialog box, click **Enroll Fingerprints**.

   The **Fingerprint Enrollment** dialog box appears.

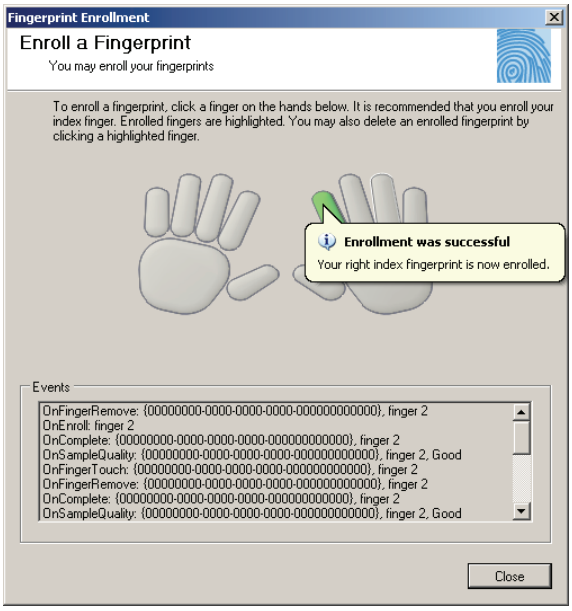


2. In the right "hand," click the index finger.

   A second **Fingerprint Enrollment** dialog box appears.



3. Using the fingerprint reader, scan your right index fingerprint.
4. Repeat step 3 until the **Enrollment was successful** message appears.



5. Click **Close**.

To verify a fingerprint

1. In the **CSharp UI Sample** dialog box, click **Verify Fingerprint**.

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

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

   In the **Verify Your Identify** dialog box, a green check mark appears over the fingerprint, which indicates that your fingerprint was verified.

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

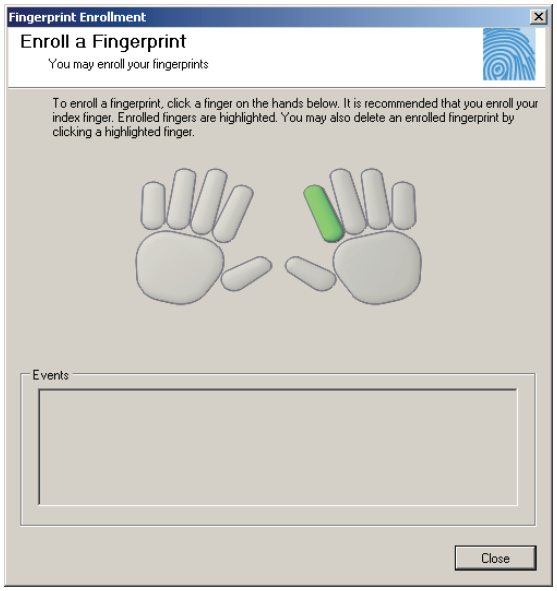
   In the **Verify Your Identify** dialog box, a red question mark appears over the fingerprint, which indicates that your fingerprint was not verified.

4. Click **Close**.

To unenroll (delete) a fingerprint

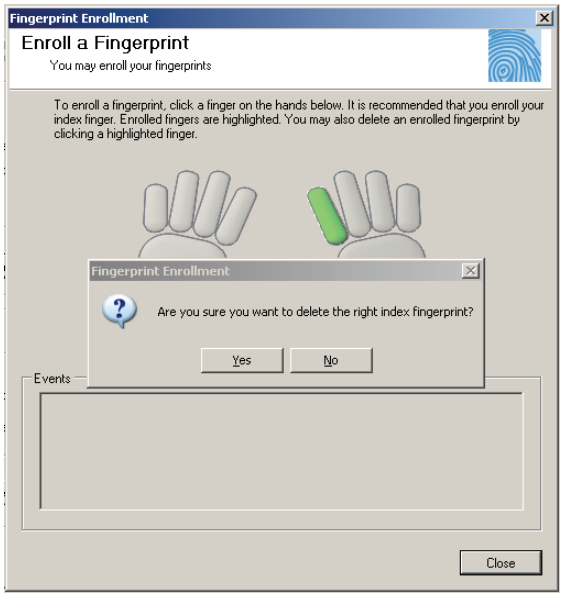
1. In the **CSharp UI Sample** dialog box, click **Enroll Fingerprints**.

   The **Fingerprint Enrollment** dialog box appears, indicating that you have enrolled your right index fingerprint.



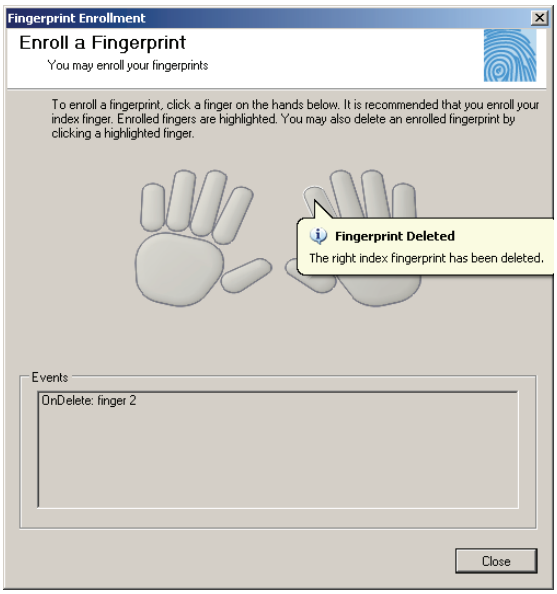
2. On the right "hand," click the green index finger.

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

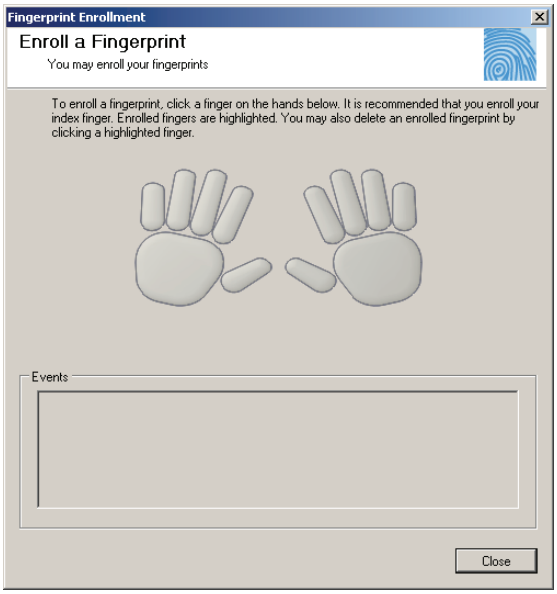


3. In the message box, click **Yes**.

   The **Fingerprint Deleted** message appears.



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



To exit the application

- In the **CSharp UI Sample** dialog box, click **Quit**.

Installation 3

This chapter contains instructions for installing the various components of the One Touch for Windows SDK: .NET 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: .NET Edition API components. This installation is located in the RTE folder. (The RTE installation is also included in the SDK installation.)

Installing the SDK

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.

To install the One Touch for Windows SDK: .NET Edition for 32-bit operating systems

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.

To install the One Touch for Windows SDK: .NET Edition for 64-bit operating systems

1. In the SDK\x64 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 for the 32-bit and 64-bit installations. The RTE files and folders, which are described in Table 2 on *page 15* for the 32-bit installation and in Table 3 on *page 16* for the 64-bit installation, are also installed on your hard disk.

Table 1. One Touch for Windows SDK: .NET Edition installed files and folders

Folder	File	Description
One Touch SDK\.NET\Bin	DPFPShrNET.dll DPFPDevNET.dll DPFPEngNET.dll DPFPVerNET.dll DPFPGuiNET.dll DPFPCtlXLib.dll DPFPCtlXTypeLibNET.dll DPFPCtlXWrapperNET.dll DPFPShrXTypeLibNET.dll	DLLs used by the One Touch for Windows: .NET Edition API
One Touch SDK\.NET\Docs	One Touch for Windows SDK .NET Developer Guide.pdf	DigitalPersona One Touch for Windows SDK: .NET Edition Developer Guide
One Touch SDK\.NET\ Samples\Visual Studio 2005\ VBNET\Enrollment	This folder contains a sample Microsoft Visual Basic 2005 project that shows how to use the One Touch for Windows: .NET Edition API for performing fingerprint enrollment and fingerprint verification.	
One Touch SDK\.NET\ Samples\Visual Studio 2005\ VBNET\UI Support	This folder contains a sample Microsoft Visual Basic 2005 project that demonstrates the functionality of the graphical user interfaces wrapped within the DPFP.Gui.Enrollment and DPFP.Gui.Verification namespaces of the One Touch for Windows: .NET Edition API.	
One Touch SDK\.NET\ Samples\Visual Studio 2005\ CSharp\Enrollment	This folder contains a sample Microsoft® Visual C#® 2005 project that shows how to use the One Touch for Windows: .NET Edition API for performing fingerprint enrollment and fingerprint verification.	
One Touch SDK\.NET\ Samples\Visual Studio 2005\ CSharp\UI Support	This folder contains a sample Microsoft Visual C# 2005 project that demonstrates the functionality of the graphical user interfaces wrapped within the DPFP.Gui.Enrollment and DPFP.Gui.Verification namespaces of the One Touch for Windows: .NET Edition API.	

Installing the Runtime Environment (RTE)

When you develop a product based on the One Touch for Windows SDK: .NET 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 105* for licensing terms.)

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

To install the One Touch for Windows: .NET Edition Runtime Environment 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 for 32-bit versions of the supported operating systems.

Table 2. One Touch for Windows: .NET 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 DPTSClnt.dll DPCrStor.dll	DLLs and executable file used by the all of the One Touch for Windows APIs
GlobalAssemblyCache	DPFPShrNET.dll DPFPDevNET.dll DPFPEngNET.dll DPFPVerNET.dll DPFPGuiNET.dll DPFPCtlXLib.dll DPFPCtlXTypeLibNET.dll DPFPCtlXWrapperNET.dll DPFPShrXTypeLibNET.dll	DLLs used by the One Touch for Windows: .NET Edition API
<*system folder*>	DPFPApi.dll DpClback.dll dpHFtrEx.dll dpHMatch.dll DPFpUI.dll	DLLs used by all of the One Touch for Windows APIs

To install the One Touch for Windows: .NET Edition Runtime Environment 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 SDK: .NET 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 DPTSClnt.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
GlobalAssemblyCache	DPFPShrNET.dll DPFPDevNET.dll DPFPEngNET.dll DPFPVerNET.dll DPFPGuiNET.dll DPFPCtlXLib.dll DPFPCtlXTypeLibNET.dll DPFPCtlXWrapperNET.dll DPFPShrXTypeLibNET.dll	DLLs used by the One Touch for Windows: .NET Edition API

Table 3. One Touch for Windows SDK: .NET Edition RTE installed files and folders, 64-bit installation *(continued)*

Folder	File	Description
<*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: .NET Edition concepts and terminology. This chapter also includes typical workflow diagrams and explanations of the One Touch for Windows: .NET Edition API functions used to perform the tasks in the workflows. For more information on fingerprint biometrics, see the "DigitalPersona White Paper: Guide to Fingerprint Recognition" included in the Docs folder of the One Touch for Windows SDK software package.

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: .NET 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 20*. Some of the tasks in these processes are done by the *fingerprint reader* and its driver; some are accomplished using One Touch for Windows: .NET 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: .NET 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: .NET 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.

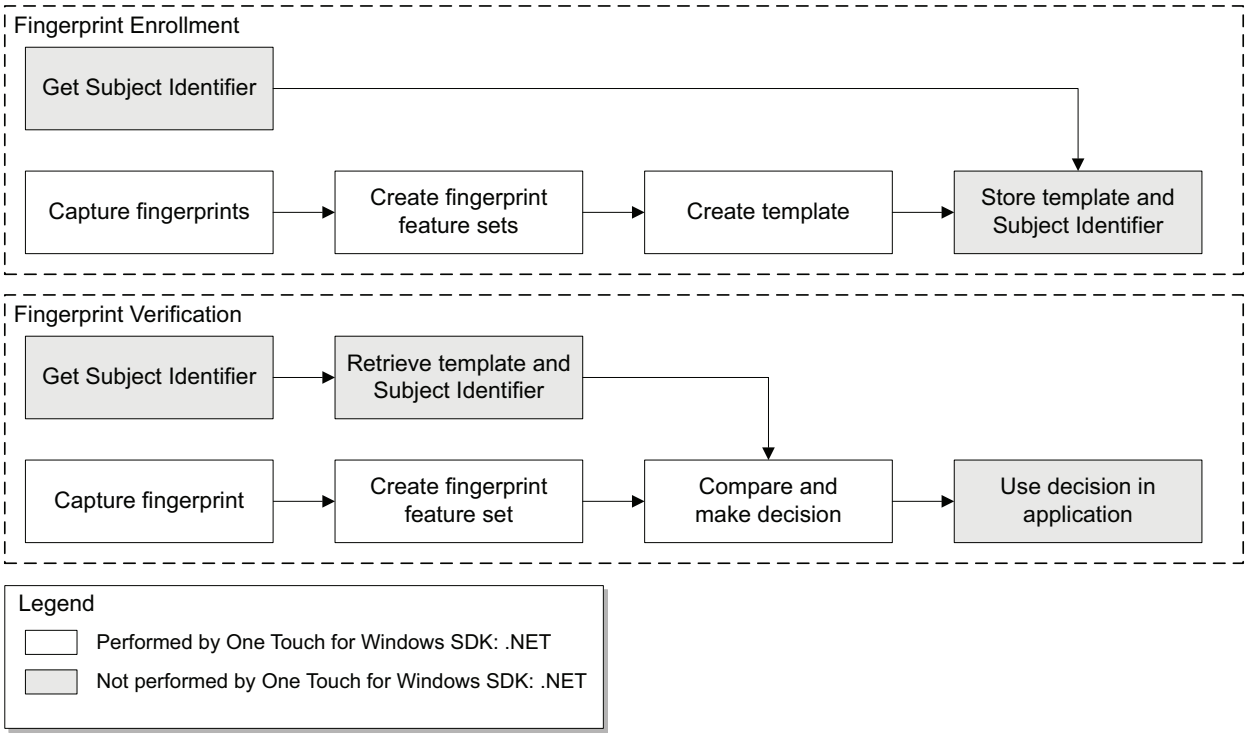


Figure 1. DigitalPersona fingerprint recognition system

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: .NET 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.

Workflows

Typical workflows are presented in this section for the following operations:

- Fingerprint enrollment
- Fingerprint enrollment with UI support
- Fingerprint verification
- Fingerprint verification with UI support
- Fingerprint data object serialization and deserialization

NOTE: Steps preceded by two asterisks (**) are done by a fingerprint reader, and steps preceded by an asterisk (*) are performed by an application.

Fingerprint Enrollment Workflow

This section contains a *typical* workflow for performing fingerprint enrollment. The workflow is illustrated in *Figure 2* and is followed by explanations of the One Touch for Windows: .NET Edition API functions used to perform the tasks in the workflow. Your application workflow may be different than the one illustrated here. For example, you could choose to create fingerprint feature sets locally and then send them to a server for enrollment.

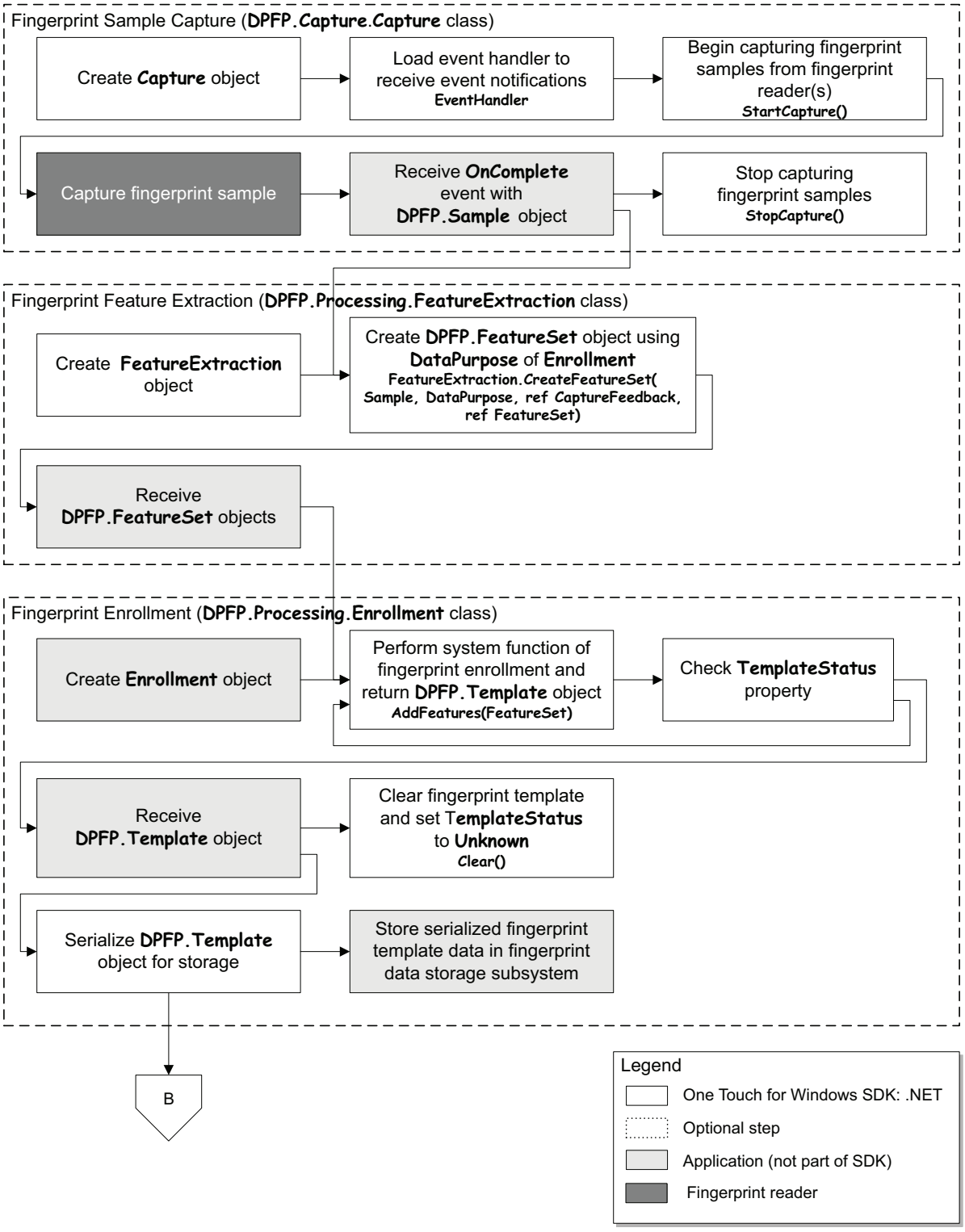


Figure 2. Typical fingerprint enrollment workflow

Fingerprint Sample Capture (DPFP.Capture.Capture Class)

1. Create a new instance of the **`DPFP.Capture.Capture`** class (*page 46*).

   IMPORTANT: You cannot change the priority or the reader(s) setting of a **`DPFP.Capture.Capture`** object after construction.

2. Load a fingerprint sample capture operation event handler for receiving event notifications by setting the **`EventHandler`** property (*page 50*).

3. Begin capturing fingerprint samples from the fingerprint reader(s) connected to a system by calling the **`StartCapture()`** method (*page 48*).

4. **Capture a fingerprint sample from a fingerprint reader.

5. *Receive the **`OnComplete`** event from the fingerprint sample capture operation event handler along with a **`DPFP.Sample`** object when the fingerprint sample is successfully captured by the fingerprint reader (*page 65*).

6. *Pass the **`DPFP.Sample`** object to the **`DPFP.Processing.FeatureExtraction.CreateFeatureSet(Sample, DataPurpose, ref CaptureFeedback, ref FeatureSet)`** method. (See step 2 in the next section.)

7. Stop capturing fingerprint samples by calling the **`StopCapture`** method (*page 49*).

Fingerprint Feature Extraction (DPFP.Processing.FeatureExtraction Class)

1. Create a new instance of the **`DPFP.Processing.FeatureExtraction`** class (*page 83*).

2. Create **`DPFP.FeatureSet`** objects by calling the **`CreateFeatureSet(Sample, DataPurpose`**, **`ref CaptureFeedback`**, **`ref FeatureSet)`** method using the value **`Enrollment`** for **`DataPurpose`** and passing the **`DPFP.Sample`** object from step 6 of the previous section (*page 83*).

3. *Pass the **`DPFP.FeatureSet`** objects created in the previous step to the **`AddFeatures`** method. (See step 2 in the next section.)

Fingerprint Enrollment (DPFP.Processing.Enrollment Class)

1. Create a new instance of the **`DPFP.Processing.Enrollment`** class (*page 85*).

2. Perform the system function of fingerprint enrollment by calling the **`AddFeatures(FeatureSet)`** method and passing the **`DPFP.FeatureSet`** objects from step 3 of the previous section (*page 85*).

3. Check the **`TemplateStatus`** property after each call to the **`AddFeatures`** method (*page 87*).

   When the **`TemplateStatus`** property returns the value **`Ready`**, a **`DPFP.Template`** object is created.

4. *Receive the **`DPFP.Template`** object.

5. Serialize the **`DPFP.Template`** object (see *Serializing a Fingerprint Data Object* on *page 34*).
6. *Store the serialized fingerprint template data in a fingerprint data storage subsystem.
7. Clear the fingerprint template and set the value of **`TemplateStatus`** to **`Unknown`** by calling the **`Clear()`** method (*page 86*).

Fingerprint Enrollment with UI Support

This section contains two *typical* workflows for performing fingerprint enrollment: one for enrolling a fingerprint and one for unenrolling (deleting) a fingerprint. The workflows are illustrated in *Figure 3* and *Figure 4* and are followed by explanations of the One Touch for Windows: .NET Edition API functions used to perform the tasks in the workflows.

Enrolling a Fingerprint

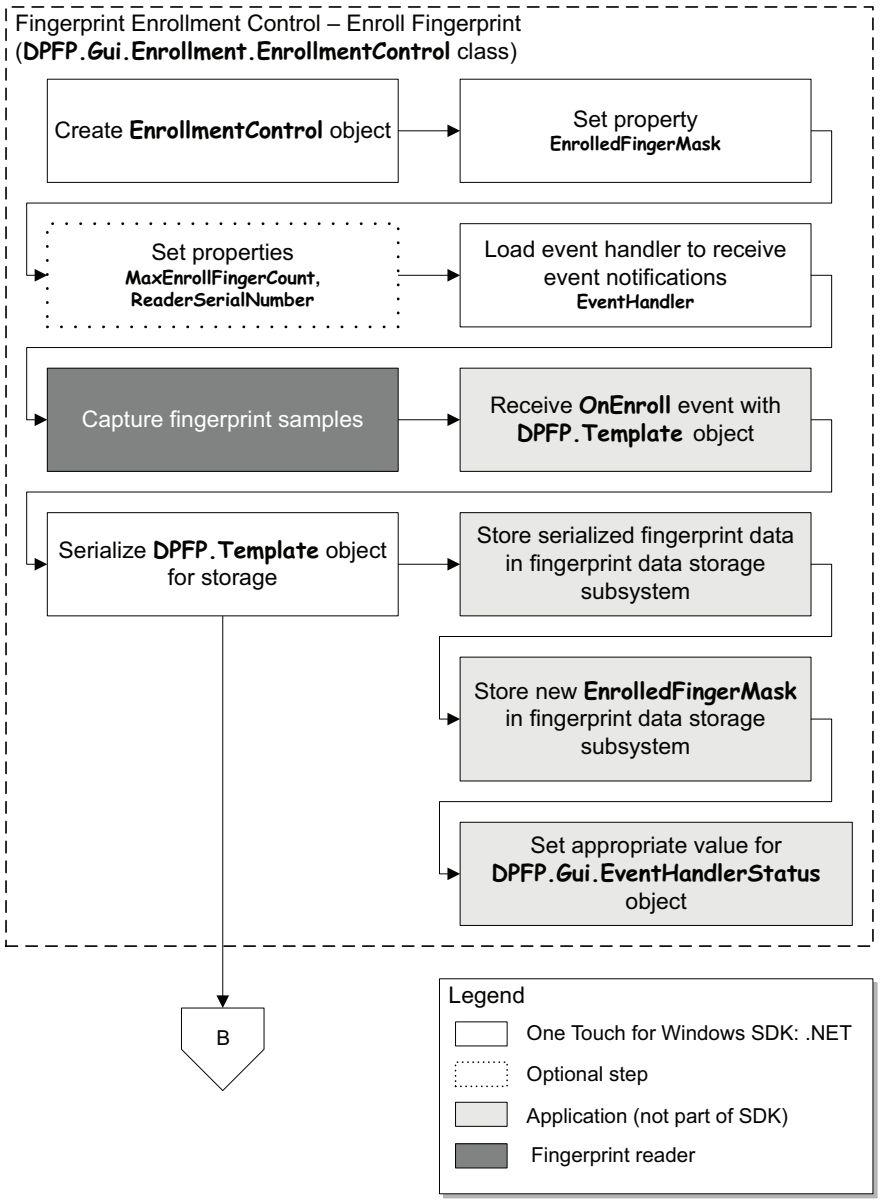


Figure 3. Typical fingerprint enrollment with UI support workflow: Enrolling a fingerprint

1. Create a new instance of the **`DPFP.Gui.Enrollment.EnrollmentControl`** class (*page 70*).
2. Set the **`EnrolledFingerMask`** property (*page 70*).
3. Optionally, set the **`MaxEnrollFingerCount`** and **`ReaderSerialNumber`** properties (*page 72* and *page 72*).
4. Load a fingerprint enrollment control event handler for receiving event notifications by setting the **`EventHandler`** property (*page 71*).
5. **Capture a predetermined number of fingerprint samples from a fingerprint reader.
6. *Receive the **`OnEnroll`** event from the fingerprint enrollment control event handler, along with the **`DPFP.Template`** object (*page 75*).
7. Serialize the **`DPFP.Template`** object (see *Serializing a Fingerprint Data Object* on *page 34*).
8. *Store the serialized fingerprint template data and the new value of the **`EnrolledFingerMask`** in a fingerprint data storage subsystem.
9. *Set the appropriate value for the **`DPFP.Gui.EventHandlerStatus`** object (*page 69*).

Unenrolling (Deleting) a Fingerprint Template

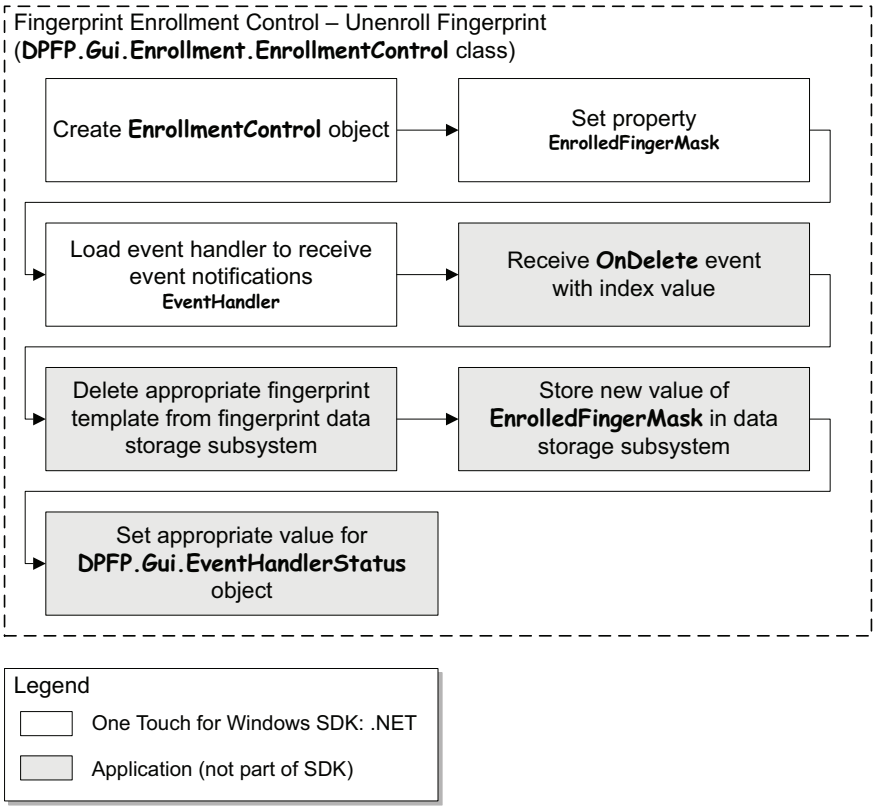


Figure 4. Typical fingerprint enrollment with UI support workflow: Unenrolling (deleting) a fingerprint

1. Create a new instance of the **`DPFP.Gui.Enrollment.EnrollmentControl`** class (*page 70*).
2. *Retrieve the value of the **`EnrolledFingerMask`** stored in the fingerprint data storage subsystem.
3. Set the **`EnrolledFingerMask`** property (*page 70*).
4. Load a fingerprint enrollment control event handler for receiving event notifications by setting the **`EventHandler`** property (*page 71*).
5. *Receive the **`OnDelete`** event from the enrollment control event handler, along with the finger index value (*page 79* and *page 74*).
6. *Delete the appropriate fingerprint template from the fingerprint data storage subsystem.
7. *Store the new value of the **`EnrolledFingerMask`** in the fingerprint data storage subsystem.
8. *Set the appropriate value for the **`DPFP.Gui.EventHandlerStatus`** object (*page 69*).

Fingerprint Verification

This section contains a *typical* workflow for performing fingerprint verification. The workflow is illustrated in *Figure 5* and is followed by explanations of the One Touch for Windows: .NET Edition API functions used to perform the tasks in the workflow.

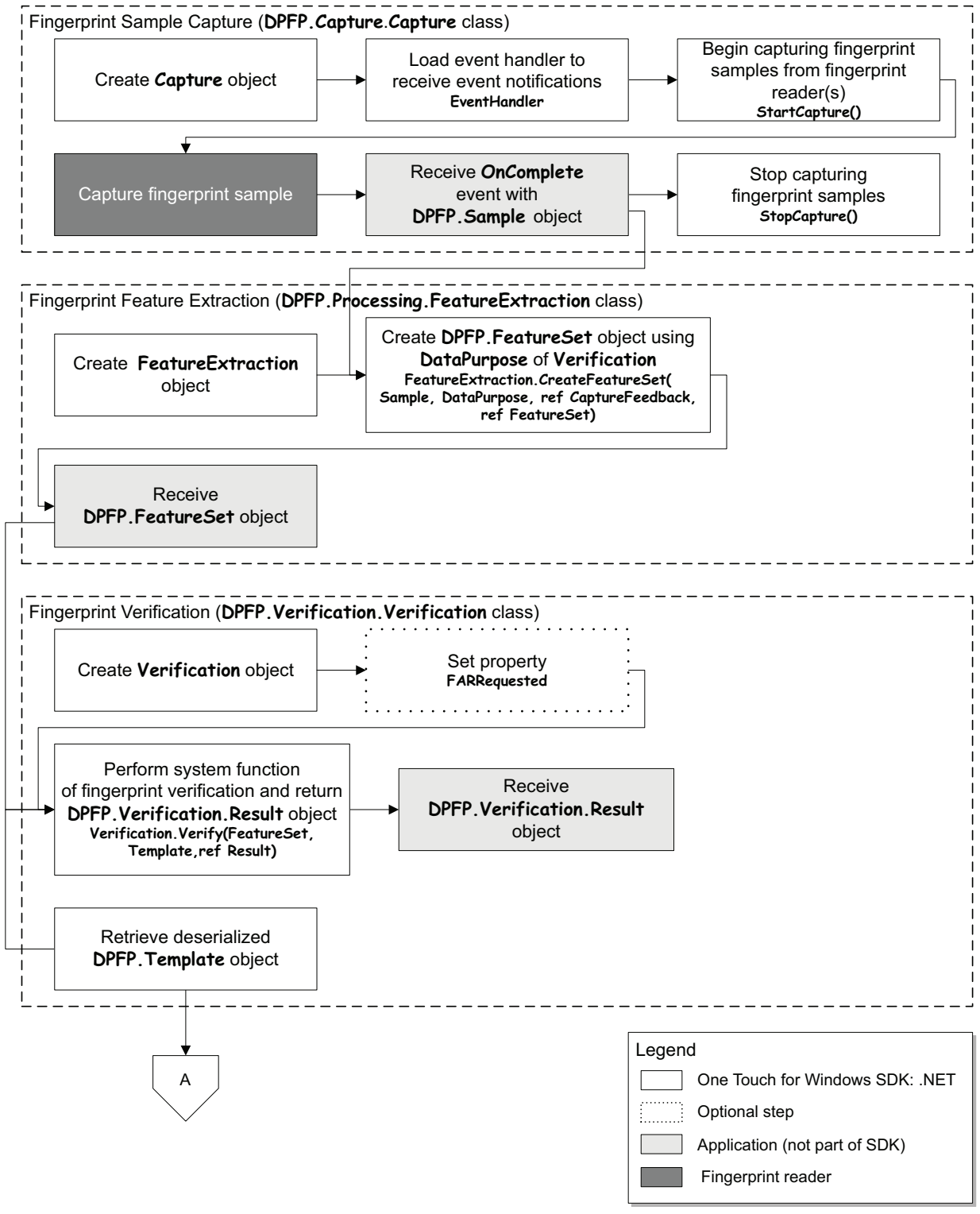


Figure 5. Typical fingerprint verification workflow

Fingerprint Sample Capture (DPFP.Capture.Capture Class)

1. Create a new instance of the **`DPFP.Capture.Capture`** class (*page 46*).

   IMPORTANT: You cannot change the priority or the reader(s) setting of a **`DPFP.Capture.Capture`** object after construction.

2. Load a fingerprint sample capture operation event handler for receiving event notifications by setting the **`EventHandler`** property (*page 50*).

3. Begin capturing fingerprint samples from the fingerprint reader(s) connected to a system by calling the **`StartCapture()`** method (*page 48*).

4. **Capture a fingerprint sample from a fingerprint reader.

5. *Receive the **`OnComplete`** event from the fingerprint sample capture operation event handler along with a **`DPFP.Sample`** object when the fingerprint sample is successfully captured by the fingerprint reader (*page 65*).

6. *Pass the **`DPFP.Sample`** object to the **`DPFP.Processing.FeatureExtraction.CreateFeatureSet(Sample, DataPurpose, ref CaptureFeedback, ref FeatureSet)`** method. (See step 2 in the next section.)

7. Stop capturing fingerprint samples by calling the **`StopCapture`** method (*page 49*).

Fingerprint Feature Extraction (DPFP.Processing.FeatureExtraction Class)

1. Create a new instance of the **`DPFP.Processing.FeatureExtraction`** class (*page 83*).

2. Create **`DPFP.FeatureSet`** objects by calling the **`CreateFeatureSet(Sample, DataPurpose`**, **`ref CaptureFeedback`**, **`ref FeatureSet)`** method using the value **`Verification`** for **`DataPurpose`** and passing the **`DPFP.Sample`** object from step 6 of the previous section (*page 83*).

3. *Pass the **`DPFP.FeatureSet`** object created in the previous step to the **`DPFP.Verification.Verification.Verify(FeatureSet, Template, ref Result)`** method. (See step 5 in the next section.)

Fingerprint Verification (DPFP.Verification.Verification Class)

1. Create a new instance of the **`DPFP.Verification.Verification`** class (*page 90*).

2. Optionally, set the **`FARRequested`** property (*page 92*). You can use this property to set or to change the value of the FAR from the default or from a specified value.

3. *Retrieve serialized fingerprint template data from the fingerprint data storage subsystem.

4. Create a **`DPFP.Template`** object from the serialized data (see *Deserializing a Serialized Fingerprint Data Object* on *page 35*).

5. Perform the system function of fingerprint verification by calling the **`Verify(FeatureSet, Template, ref Result)`** method and passing the **`DPFP.Template`** object created in the previous step and the **`DPFP.FeatureSet`** object from step 3 of the previous section (*page 91*).

6. *Receive the **`DPFP.Verification.Result`** object, which provides the comparison decision of match or non-match (*page 93*).

Fingerprint Verification with UI Support

This section contains a *typical* workflow for performing fingerprint verification with UI support. The workflow is illustrated in *Figure 6* and is followed by explanations of the One Touch for Windows: .NET Edition API functions used to perform the tasks in the workflow.

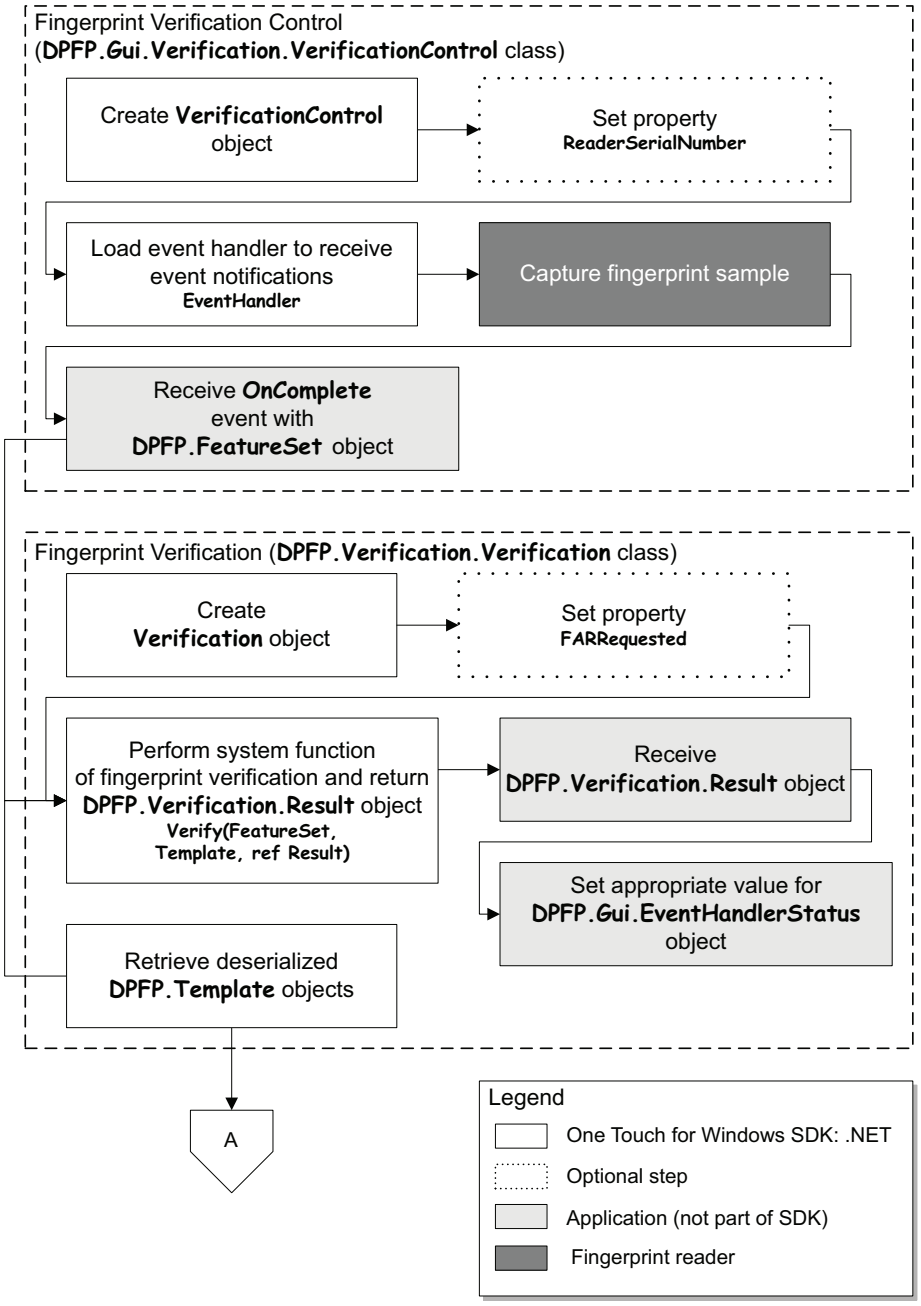


Figure 6. Typical fingerprint verification with UI support workflow

Fingerprint Verification Control (DPFP.Gui.Verification.VerificationControl Class)

1. Create a new instance of the **`DPFP.Gui.Verification.VerificationControl`** class (*page 80*).
2. Optionally, set the **`ReaderSerialNumber`** property (*page 81*).
3. Load a fingerprint verification control event handler for receiving event notifications by setting the **`EventHandler`** property (*page 81*).
4. **Capture a fingerprint sample from a fingerprint reader.
5. Receive the **`OnComplete`** event from the fingerprint verification control event handler along with the **`DPFP.FeatureSet`** object (*page 82*).

Fingerprint Verification (DPFP.Verification.Verification Class)

1. Create a new instance of the **`DPFP.Verification.Verification`** class (*page 90*).
2. Optionally, set the **`FARRequested`** property (*page 92*). You can use this property to set or to change the value of the FAR from the default or from a specified value.
3. *Retrieve serialized fingerprint template data from the fingerprint data storage subsystem.
4. Create a **`DPFP.Template`** object from serialized data (see *Deserializing a Serialized Fingerprint Data Object* on *page 35*).
5. Perform the system function of fingerprint verification by calling the **`Verify(FeatureSet, Template, ref Result)`** method and passing the **`DPFP.Template`** and **`DPFP.FeatureSet`** objects (*page 91*).
6. *Receive the **`DPFP.Verification.Result`** object, which provides the comparison decision of match or non-match (*page 93*).
7. *Set the appropriate value for the **`DPFP.Gui.EventHandlerStatus`** object (*page 69*).

Fingerprint Data Object Serialization/Deserialization

This section contains two workflows: one for serializing a fingerprint data object and one for deserializing a serialized fingerprint data object. The workflows are illustrated in *Figure 7* and *Figure 8* and are followed by explanations of the One Touch for Windows: .NET Edition API functions used to perform the tasks in the workflows.

Serializing a Fingerprint Data Object

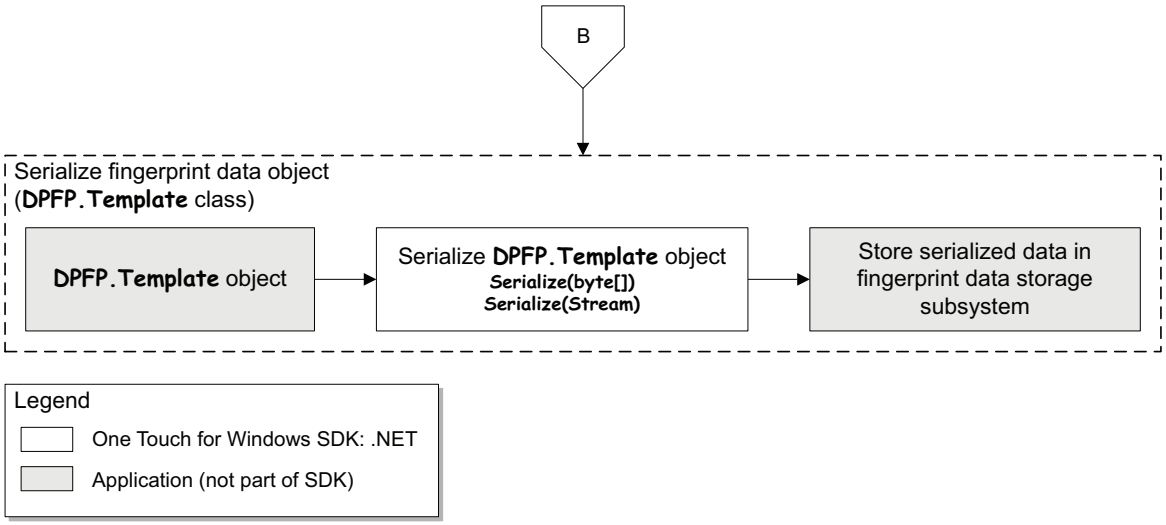


Figure 7. Fingerprint data object serialization workflow: `DPFP.Template` object

1. Begin with a `DPFP.Template` object. (See *DPFP.Template Class* on *page 43* for more information on how a `DPFP.Template` object is constructed or supplied).
2. Serialize the `DPFP.Template` object by calling the `DPFP.Template.Serialize(byte[])` or `DPFP.Template.Serialize(Stream)` method (*page 38* and *page 39*).
3. *Store the serialized fingerprint template data in a fingerprint data storage subsystem.

Deserializing a Serialized Fingerprint Data Object

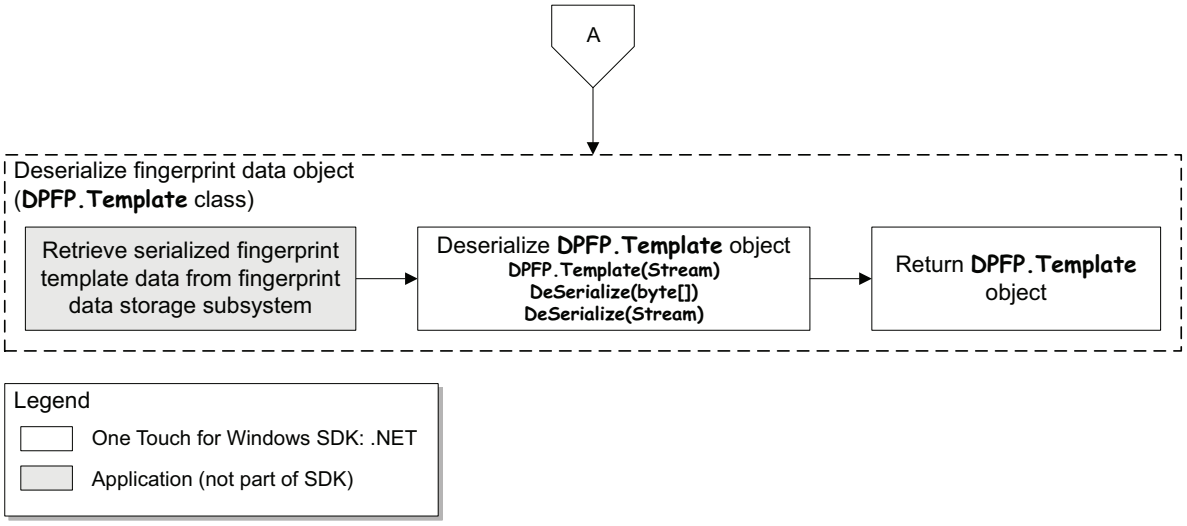


Figure 8. Deserialization of serialized fingerprint data object workflow: **`DPFP.Template`** object

1. *Retrieve serialized fingerprint template data from a fingerprint data storage subsystem.
2. Deserialize a **`DPFP.Template`** object by calling the **`DPFP.Template(Stream)`** constructor, or the **`DPFP.Template.DeSerialize(byte[])`** or **`DPFP.Template.DeSerialize(Stream)`** method (*page 38*, *page 39*, and *page 40*, respectively).
3. Return a **`DPFP.Template`** object.

API Reference 5

This chapter defines the components for developing applications that incorporate the functionality of the One Touch for Windows: .NET Edition API.

IMPORTANT: All of the read/write properties of the One Touch for Windows: .NET Edition API components are optional. If you do not set one of these properties, the default value is automatically used. When deciding whether to set a property, be aware that DigitalPersona may change the default values at any time without notice. If you want your application's functionality to remain consistent, you should set the properties accordingly.

Exceptions

The One Touch for Windows SDK: .NET Edition extensively employs the .NET exception mechanism to signify a failed operation. Since the SDK sometimes returns a return code/value on function invocation, it is strongly recommended that you use effective exception handling when calling the API.

All SDK-specific exceptions are packaged within the **`DPFP.Error.SDKException`** class and should be trapped accordingly. The *Exceptions* sections in this API reference list delivery possibilities for each method of each SDK object.

The **`DPFP.Error.SDKException`** class extends ApplicationException by adding the **`ErrorCode`** property defined on *page 43*, which returns an enumerated value of **`DPFP.Error.ErrCodes`** (*page 44*).

The **`DPFP.Error.SDKException`** class also provides more information through the following properties:

- **`Message`** — a string detailing the nature of the exception
- **`InnerException`** — a System.Exception further detailing the nature of the exception

Components

The One Touch for Windows: .NET Edition API includes the components defined in the remainder of this chapter. Use the following list to quickly locate a component by name, by page number, or by description.

Component	Page	Description
Shared	*38*	This component is wrapped within the **`DPFP`** namespace. The members of this component are shared by other components of the One Touch for Windows: .NET Edition API. This component is always installed.
Capture	*46*	This component is wrapped within the **`DPFP.Capture`** namespace. The members of this component ■ Capture fingerprint samples from fingerprint readers ■ Fire events from fingerprint readers ■ Provide information about the fingerprint readers connected to a system ■ Convert a fingerprint sample to an image
GUI	*69*	This component is wrapped within the **`DPFP.Gui`** namespace, which provides graphical user interfaces for performing fingerprint enrollment and fingerprint verification operations and event handler status feedback. The **`DPFP.Gui.Enrollment`** and **`DPFP.Gui.Verification`** namespaces are wrapped within the **`DPFP.Gui`** namespace.
Processing	*83*	This component is wrapped within the **`DPFP.Processing`** namespace. The members of this component provide methods and properties for ■ Creating fingerprint feature sets for the purpose of enrollment or verification ■ Performing the system function of fingerprint enrollment by creating fingerprint templates
Verification	*90*	This component is wrapped within the **`DPFP.Verification`** namespace. The members of this component provide a method and properties for ■ Performing the system function of fingerprint verification ■ Returning and retrieving the false accept rate (FAR) ■ Returning the results of the fingerprint verification operation

Shared Component

The shared component is wrapped within the `DPFP` namespace. The members of this component are shared by other components of the One Touch for Windows: .NET Edition API. This component is always installed.

DPFP.Data Class

Represents the data that is common to all *fingerprint data objects*. The `Data` class also provides methods and properties for serializing and deserializing fingerprint data objects.

Default Constructors

Data()

A base class to all of the fingerprint data objects: `DPFP.FeatureSet`, `DPFP.Sample`, and `DPFP.Template`.

Syntax

```
public Data()
```

Data(Stream)

Constructs a data object from a given stream.

Syntax

```
public Data(Stream DataStream)
```

Parameter

`DataStream`	Data stream to deserialize

Public Methods

Serialize(ref byte[])

Serializes a data object and returns it as an array of bytes.

Syntax

```
public void Serialize(
  ref byte[] ArrayOfBytes
)
```

Parameter

`ArrayOfBytes`	**Array of bytes** that receives and returns a serialized data object

Exceptions

DPFP.Error.ErrorCodes	Message	Reason
Internal	Bad Serialization	Serialization failed, probably due to memory limitations or bad arguments.

DeSerialize (byte[])

Deserializes a data object returned by the `Serialize` method.

Syntax

```
public void DeSerialize(
  byte[] ArrayOfBytes
)
```

Parameters

`ArrayOfBytes`	**Array of bytes** that contains a deserialized data object

Exceptions

DPFP.Error.ErrorCodes	Message	Reason
Internal	Bad DeSerialization	Deserialization failed, probably due to memory limitations or bad arguments.

Serialize(Stream)

Serializes a data object to a stream.

Syntax

```
public Stream Serialize(Stream DataStream)
```

Parameter

`DataStream`	Data stream

Exception

DPFP.Error.ErrorCodes	Message	Reason
Internal	Bad Serialization	Serialization failed, probably due to memory limitations or bad arguments.

DeSerialize(Stream)

Deserializes a data object returned by the `Serialize` method.

Syntax

```
public Stream DeSerialize(Stream DataStream)
```

Parameter

`DataStream`	Data stream

Exception

DPFP.Error.ErrorCodes	Message	Reason
Internal	Bad DeSerialization	Deserialization failed, probably due to memory limitations or bad arguments.

Public Properties

Bytes

Returns embedded raw data.

Syntax

```
public byte[] Bytes
```

Parameter

`Bytes`	**Array of bytes** that receives the embedded raw data

This property is read-only.

Size

Returns the size of the embedded raw data, in bytes.

Syntax

```
public int Size
```

Parameter

`Size`	**Int** that receives the size of the embedded raw data, in bytes

This property is read-only.

DPFP.FeatureSet Class

Represents a fingerprint feature set. The `DPFP.FeatureSet` object is supplied in the `FeatureSet` parameter of the `CreateFeatureSet` method (*page 83*) and by the `OnComplete` event of the fingerprint verification control event handler (*page 82*).

NOTE: The `DPFP.FeatureSet` class inherits all public methods and properties from the `DPFP.Data` class.

Public Constructors

FeatureSet()

Constructs a `DPFP.FeatureSet` object.

Syntax

```
public FeatureSet()
```

FeatureSet(Stream)

Constructs a `DPFP.FeatureSet` object from a given stream.

Syntax

```
public FeatureSet(Stream DataStream)
```

Parameter

`DataStream`	Data stream to deserialize

Exception

DPFP.Error.ErrorCodes	Message	Reason
Internal	Bad DeSerialization	Deserialization failed, probably due to memory limitations or bad arguments.

DPFP.Sample Class

Represents a fingerprint sample captured from a fingerprint reader. The `DPFP.Sample` object is supplied by the `OnComplete` event of the fingerprint sample capture operation event handler (*page 65*).

NOTE: The `DPFP.Sample` class inherits all public methods and properties from the `DPFP.Data` class.

Public Constructors

Sample()

Constructs a `DPFP.Sample` object.

Syntax

```
public Sample()
```

Sample(Stream)

Constructs a `DPFP.Sample` object from a given stream.

Syntax

```
public Sample(Stream DataStream)
```

Parameter

`DataStream`	Data stream to deserialize

Exception

DPFP.Error.ErrorCodes	Message	Reason
Internal	Bad DeSerialization	Deserialization failed, probably due to memory limitations or bad arguments.

DPFP.Template Class

Represents a fingerprint template. The **`DPFP.Template`** object is supplied by the **`Template`** property (*page 87*) and by the **`OnEnroll`** event of the fingerprint enrollment control event handler (*page 75*).

NOTE: The **`DPFP.Template`** class inherits all public methods and properties from the **`DPFP.Data`** class.

Public Constructors

Template()

Constructs a **`DPFP.Template`** object.

Syntax

```
public Template()
```

Template(Stream)

Constructs a **`DPFP.Template`** object from a given stream.

Syntax

```
public Template(Stream DataStream)
```

Parameter

`DataStream`	Data stream to deserialize

Exception

DPFP.Error.ErrorCodes	Message	Reason
Internal	Bad DeSerialization	Deserialization failed, probably due to memory limitations or bad arguments.

DPFP.Error.SDKException Class

Provides SDK-specific exceptions.

Public Property

ErrorCode

Returns an embedded error code.

Syntax

```
public ErrorCodes ErrorCode
```

Possible Values

ErrorCode	Enumeration that receives one of the values from **DPFP.Error.ErrorCodes**

Public Enumeration

ErrorCodes

Defines the error codes returned by the **SDKException** class.

Syntax

```
public enum ErrorCodes
{
  Success = 0,
  NotInitialized = -1,
  InvalidParameter = -2,
  NotImplemented = -3,
  IO = -4,
  NoMemory = -7,
  Internal = -8,
  BadSetting = -9,
  UnknownDevice = -10,
  InvalidBuffer = -11,
  FeatureSetTooShort = -16,
  InvalidContext = -17,
  InvalidFeatureSetType = -29,
  InvalidFeatureSet = -32,
  Unknown = -33
}
```

Members

Success	The function succeeded.
NotInitialized	Some Engine components are missing or inaccessible.
InvalidParameter	One or more parameters are not valid.
IO	A generic I/O file error occurred.
NoMemory	There is not enough memory to perform the action.

`Internal`	An unknown internal error occurred.
`BadSetting`	Initialization settings are corrupted.
`UnknownDevice`	The requested device is not known.
`InvalidBuffer`	A buffer is not valid.
`FeatureSetTooShort`	The specified fingerprint feature set or fingerprint template buffer size is too small.
`InvalidContext`	The given context is not valid.
`InvalidFeatureSetType`	The feature set purpose is not valid.
`InvalidFeatureSet`	Decrypted fingerprint features are not valid. Decryption may have failed.
`Unknown`	An unknown exception occurred.

Remarks

The members of this enumeration are used by the **`DPFP.Error.ErrorCode`** property (*page 43*).

Library

DPFPShrNet.dll

Capture Component

The capture component is wrapped within the `DPFP.Capture` namespace. The members of this component

- Capture fingerprint samples from fingerprint readers
- Fire events from fingerprint readers
- Provide information about the fingerprint readers connected to a system
- Convert a fingerprint sample to an image

DPFP.Capture.Capture Class

Captures a fingerprint sample from a particular fingerprint reader or from all of the fingerprint readers connected to a system and may specify the priority for the capture operation.

Public Constructors

Capture(String, Priority)

Initializes a new instance of the `Capture` class for capturing a fingerprint sample from a particular fingerprint reader using its serial number and specifies any valid value for the priority of the capture operation.

Syntax

```
public Capture(String ReaderSerialNumber, Priority CapturePriority)
```

Parameters

`ReaderSerialNumber`	**String** that contains a fingerprint reader serial number
`CapturePriority`	Enumeration that specifies one of the values from `DPFP.Capture.Priority` (*page 64*)

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.

Capture(String)

Initializes a new instance of the `Capture` class for capturing a fingerprint sample from a particular fingerprint reader using its serial number and assigns `Normal` priority to the capture operation.

Syntax

```
public Capture(String ReaderSerialNumber)
```

Parameter

`ReaderSerialNumber`	**String** that contains a fingerprint reader serial number

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.

Capture(Priority)

Initializes a new instance of the `Capture` class for capturing a fingerprint sample from all of the fingerprint readers connected to a system and specifies any valid value for the priority of the capture operation.

Syntax

```
public Capture(Priority CapturePriority)
```

Parameter

`CapturePriority`	Enumeration that specifies one of the values from `DPFP.Capture.Priority` (*page 64*)

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.

Capture()

Initializes a new instance of the `Capture` class for capturing a fingerprint sample from any of the fingerprint readers connected to a system and assigns `Normal` priority to the capture operation.

Syntax

```
public Capture()
```

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.

Public Methods

StartCapture()

Begins capturing a fingerprint sample from a fingerprint reader. A call to this method is asynchronous and returns immediately. If a fingerprint sample capture operation event handler was loaded through the `EventHandler` property, the application receives events from the fingerprint reader (*page 50*). Every call to the `StartCapture()` method must be paired with a call to the `StopCapture()` method.

Syntax

```
public void StartCapture()
```

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.
UnknownDevice	Failed to access the reader	The specified reader ID is either not valid or does not exist.
Internal	Failed to create acquisition	The fingerprint sample capture operation failed to initialize.
Internal	Failed to start acquisition	The fingerprint sample capture operation failed to start.

StopCapture()

Stops the fingerprint sample capture operation started with a call to the **`StartCapture()`** method.

Syntax

```
public void StopCapture()
```

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.

Public Properties

Priority

Returns a value that specifies the priority of a fingerprint sample capture operation.

Syntax

```
public Priority Priority
```

Possible Values

`Priority`	Enumeration that receives one of the values from **`DPFP.Capture.Priority`** (*page 64*)

This property is read-only and may also be set through construction.

ReaderSerialNumber

Returns the serial number of a fingerprint reader that captures a fingerprint sample.

Syntax

```
public string ReaderSerialNumber
```

Possible Values

`ReaderSerialNumber`	**String** that receives a fingerprint reader serial number

This property is read-only and may also be set through construction.

EventHandler

Loads a fingerprint sample capture operation event handler. Set this property to **`null`** to clear all registered event handlers.

IMPORTANT: At least one event handler should be loaded to receive events.

Syntax

```
public EventHandler EventHandler
```

Possible Values

`EventHandler`	A **`DPFP.Capture.EventHandler`** object (*page 65*)

This property is write-only.

DPFP.Capture.ReaderDescription Class

Provides information about a particular fingerprint reader, such as its technology or serial number.

Public Constructors

ReaderDescription(Guid)

Initializes a new instance of the **`ReaderDescription`** class using a fingerprint reader's device GUID.

Syntax

```
public ReaderDescription(Guid DeviceGUID)
```

Parameter

`DeviceGUID`	Variable that contains a fingerprint reader device GUID

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.
UnknownDevice	Failed to access the reader	The specified reader ID is either not valid or does not exist.

ReaderDescription(String)

Initializes a new instance of the **`ReaderDescription`** class using a fingerprint reader's serial number.

Syntax

```
public ReaderDescription(String ReaderSerialNumber)
```

Parameter

`ReaderSerialNumber`	**String** that contains a fingerprint reader serial number

Exceptions

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possible due to misconfiguration, incompatible binaries, or security/ memory limitations.
InvalidParameter	Invalid index or serial number	The format of the specified serial number is not valid.
UnknownDevice	Failed to access the device	The specified reader ID is either not valid or does not exist.

Public Properties

FirmwareRevision

Returns the firmware revision number of a fingerprint reader.

Syntax

```
public ReaderVersion FirmwareVersion
```

Possible Values

`FirmwareRevision`	Variable that receives a fingerprint reader firmware revision number

This property is read-only.

HardwareRevision

Returns the hardware revision number of a fingerprint reader.

Syntax

```
public ReaderVersion HardwareVersion
```

Possible Values

`HardwareVersion`	Variable that receives a fingerprint reader hardware revision number

This property is read-only.

Language

Returns the fingerprint reader language.

Syntax

```
public uint Language
```

Possible Values

`Language`	**Uint** the receives the fingerprint reader language. The value of **`Language`** is always 0x409, which is English.

This property is read-only.

ImpressionType

Returns a value that specifies the fingerprint reader impression type, for example, swipe reader or touch (area) reader.

Syntax

```
public ReaderImpressionType ImpressionType
```

Possible Values

`ImpressionType`	Enumeration that receives one of the values from **`DPFP.Capture.ReaderImpressionType`** (*page 54*)

This property is read-only.

ProductName

Returns the product name of a fingerprint reader, for example, "U.are.U."

Syntax

```
public String ProductName
```

Possible Values

`ProductName`	**String** that receives the fingerprint reader product name

This property is read-only.

SerialNumber

Returns the serial number of a fingerprint reader.

Syntax

```
public String SerialNumber
```

Possible Values

`SerialNumber`	**String** the receives the fingerprint reader serial number

This property is read-only.

SerialNumberType

Returns a value that specifies the type of fingerprint reader serial number.

Syntax

```
public ReaderSerialNumberType SerialNumberType
```

Possible Values

`SerialNumberType`	Enumeration that receives one of the values from `DPFP.Capture.SerialNumberType` (*page 56*)

This property is read-only.

Technology

Returns a value that specifies the fingerprint reader technology.

Syntax

```
public ReaderTechnology Technology
```

Possible Values

`Technology`	Enumeration that receives one of the values from **`DPFP.Capture.ReaderTechnology`** (*page 55*)

This property is read-only.

Vendor

Returns the vendor name for a fingerprint reader, for example, "DigitalPersona, Inc."

Syntax

```
public String Vendor
```

Possible Values

`Vendor`	**String** the receives the fingerprint reader vendor name

This property is read-only.

Public Enumerations

ReaderImpressionType

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

Syntax

```
public enum ReaderImpressionType
{
  Unknown = 0,
  Swipe,
  Area
};
```

Members

`Unknown`	A fingerprint reader for which the modality is not known.
`Swipe`	A swipe fingerprint reader.
`Area`	An area (touch) sensor fingerprint reader.

Remarks

The members of this enumeration are used by the **`DPFP.Capture.Capture.ImpressionType`** property (*page 52*).

ReaderTechnology

Defines the fingerprint reader technology.

Syntax

```
public enum ReaderTechnology
{
  Unknown = 0,
  Optical,
  Capacitive,
  Thermal,
  Pressure
};
```

Members

`Unknown`	A fingerprint reader for which the technology is not known.
`Optical`	An optical fingerprint reader.
`Capacitive`	A capacitive fingerprint reader.
`Thermal`	A thermal fingerprint reader.
`Pressure`	A pressure fingerprint reader.

Remarks

The members of this enumeration are used by the **`DPFP.Capture.Capture.Technology`** property (*page 54*).

SerialNumberType

Defines whether a fingerprint reader serial number persists after reboot.

Syntax

```
public enum ReaderSerialNumberType
{
  Persistent = 0,
  Volatile
};
```

Members

`Persistent`	A persistent serial number provided by the hardware.
`Volatile`	A volatile serial number provided by the software.

Remarks

The members of this enumeration are used by the **`DPFP.Capture.Capture.SerialNumberType`** property (*page 53*).

DPFP.Capture.ReadersCollection Class

Provides information about all of the fingerprint readers connected to a system.

Public Constructor

ReadersCollection()

Initializes a new instance of the **`ReadersCollection`** class for enumerating all of the fingerprint readers connected to a system.

Syntax

```
public ReadersCollection()
```

Exceptions

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.
Internal	Failed to enumerate readers	The device enumeration failed.

Public Method

Refresh()

Clears and re-enumerates a **`ReadersCollection`** object.

Syntax

```
public void Refresh()
```

Exceptions

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.
Internal	Failed to enumerate readers	The device enumeration failed.

Public Indexers

ReaderDescription this[Guid]

Returns a specific **`ReaderDescription`** object using its device GUID.

Syntax

```
public ReaderDescription this[Guid ReaderSerialNumber]
```

Parameter

`ReaderSerialNumber`	Variable that contains a fingerprint reader device GUID

Exception

DPFP.Error.ErrorCodes	Message	Reason
UnknownDevice	Failed to access the reader	The specified reader ID is either not valid or does not exist.

ReaderDescription this[int]

Returns a specific fingerprint reader using its index.

Syntax

```
public ReaderDescription this[int Index]
```

Parameter

`Index`	**Int** that contains a fingerprint reader index

Exception

DPFP.Error.ErrorCodes	Message	Reason
InvalidParameter	Invalid index or serial number	The specified index is not within the valid range.

ReaderDescription this[string]

Returns a specific fingerprint reader using its serial number.

Syntax

```
public ReaderDescription this[string ReaderSerialNumber]
```

Parameter

`ReaderSerialNumber`	**String** that contains a fingerprint reader serial number

Exceptions

DPFP.Error.ErrorCodes	Message	Reason
InvalidParameter	Invalid index or serial number	The format of the specified serial number is not valid.
UnknownDevice	Failed to access the device	The specified reader ID is either not valid or does not exist.

DPFP.Capture.ReaderVersion Class

Represents information about the fingerprint reader version.

Public Constructor

ReaderVersion(uint, uint, uint)

Initializes a new instance of the `ReaderVersion` class for providing the structure of the fingerprint reader version number.

Syntax

```
public ReaderVersion(uint Build, uint Major, uint Minor)
```

Parameters

`Build`	**Uint** that contains a build number
`Major`	**Uint** that contains a major number
`Minor`	**Uint** that contains a minor number

Public Method

ToString()

Converts a `DPFP.Capture.ReaderVersion` object to a string representation.

Syntax

```
public string ToString()
```

Parameters

`ToString`	**String** that contains a version number

Public Properties

Build

Returns the build number of the fingerprint reader version.

Syntax

```
public uint Build
```

Possible Values

`Build`	**Uint** that receives the build number

This property is read-only.

Major

Returns the major number of the fingerprint reader version.

Syntax

```
public uint Major
```

Possible Values

`Major`	**Uint** that receives the major number

This property is read-only.

Minor

Returns the minor number of the fingerprint reader version.

Syntax

```
public uint Major
```

Possible Values

`Minor`	**Uint** that receives the minor number

This property is read-only.

DPFP.Capture.SampleConversion Class

Provides methods for converting a fingerprint sample to an image in either bitmap image file format or ANSI 381 format.

Public Constructor

SampleConversion()

Initializes a new instance of the `SampleConversion` class for converting a fingerprint sample to an image.

Syntax

```
public SampleConversion()
```

Exceptions

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.

Public Methods

ConvertToANSI381(Sample, ref byte[])

Converts a fingerprint sample to an image in ANSI 381 format.

Syntax

```
public byte[] ConvertToANSI381(
  Sample Sample,
  ref byte[] ANSI
)
```

Parameters

`Sample`	A `DPFP.Sample` object (*page 42*)
`ANSI`	**Array of bytes** that receives and contains an image in ANSI 381 format

Return Value

Returns an array of bytes that receives and contains an image in ANSI 381 format.

Exceptions

DPFP.Error.ErrorCodes	Message	Reason
InvalidParameter	Invalid Parameter	The arguments provided for this function are either null or contain no valid data.
Internal	Failed to export ANSI 381 image	The conversion to ANSI 381 format failed.

ConvertToPicture(Sample, ref Bitmap)

Converts a fingerprint sample to bitmap image file format.

Syntax

```
public Bitmap ConvertToPicture(
  Sample Sample,
  ref Bitmap Bitmap
)
```

Parameters

Sample	A **DPFP.Sample** object (*page 42*)
Bitmap	Variable that receives and contains an image in bitmap file format and scales the image to a specified bitmap size, if provided

Return Value

Returns a bitmap that receives and contains an image in bitmap file format.

Exceptions

DPFP.Error.ErrorCodes	Message	Reason
InvalidParameter	Invalid Parameter	The arguments provided for this function are either null or contain no valid data.
Internal	Failed to export ANSI 381 image	The conversion to bitmap image file format failed.

DPFP.Capture.CaptureFeedback Enumeration

Defines the values that provide feedback about a fingerprint sample capture operation.

Syntax

```
public enum CaptureFeedback
{
  Good = 0,
  None,
  TooLight,
  TooDark,
  TooNoisy,
  LowContrast,
  NotEnoughFeatures,
  NoCentralRegion,
  NoFinger,
  TooHigh,
  TooLow,
  TooLeft,
  TooRight,
  TooStrange,
  TooFast,
  TooSkewed,
  TooShort,
  TooSlow,
  TooSmall
}
```

Members

Good	The fingerprint sample is of decent quality.
None	The fingerprint sample is missing, or was not received.
TooLight	The fingerprint sample is too light.
TooDark	The fingerprint sample is too dark
TooNoisy	The fingerprint sample is too noisy.
LowContrast	The fingerprint sample contrast is too low.
NotEnoughFeatures	The fingerprint sample does not contain enough information.
NoCentralRegion	The fingerprint sample is not centered.

`NoFinger`	The scanned object is not a finger.
`TooHigh`	The finger was too high on the swipe sensor.
`TooLow`	The finger was too low on the swipe sensor.
`TooLeft`	The finger was too close to the left border of the swipe sensor.
`TooRight`	The finger was too close to the right border of the swipe sensor.
`TooStrange`	The scan looks strange.
`TooFast`	The finger was swiped too quickly.
`TooSkewed`	The fingerprint sample is too skewed.
`TooShort`	The fingerprint sample is too short.
`TooSlow`	The finger was swiped too slowly.
`TooSmall`	The size of the fingerprint sample is too small.

Remarks

The members of this enumeration are used by the **`DPFP.Processing.CreateFeatureSet`** method (*page 83*) and by the **`OnSampleQuality`** event of the fingerprint sample capture event handler (*page 68*).

DPFP.Capture.Priority Enumeration

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

Syntax

```
public enum Priority
{
  Low = 0,
  Normal,
  High
};
```

Members

`Low`	Low priority. An application uses this priority to acquire events from the fingerprint reader only if there are no subscribers with high or normal priority. Only one subscriber with this priority is allowed.
`Normal`	Normal priority. An application uses this priority to acquire events from the fingerprint reader only if the operation runs in a foreground process. Multiple subscribers with this priority are allowed.
`High`	High priority. (RESERVED. For internal use only.) A subscriber uses this priority to acquire events from the fingerprint reader exclusively. Only one subscriber with this priority is allowed. Note that under Windows Vista and later, the subscriber must be the system account or a member of the Administrators Group

Remarks

The members of this enumeration are used by the **`DPFP.Capture.Priority`** property (*page 49*).

DPFP.Capture.EventHandler Interface

Defines the fingerprint sample capture operation events.

Syntax

```
public interface EventHandler
{
  void OnComplete(Object, String, Sample);
  void OnFingerGone(Object, String);
  void OnFingerTouch(Object, String);
  void OnReaderConnect(Object, String);
  void OnReaderDisconnect(Object, String);
  void OnSampleQuality(Object, String, CaptureFeedback);
}
```

Events

OnComplete(Object, String, Sample)

Fires when a fingerprint sample is successfully captured by a fingerprint reader.

Syntax

```
public void OnComplete(Object Capture,
  String ReaderSerialNumber,
  Sample Sample);
```

Parameters

`Capture`	A `DPFP.Capture.Capture` object (*page 46*)
`ReaderSerialNumber`	**String** that contains the reader ID of the fingerprint reader from which the fingerprint sample was captured
`Sample`	A `DPFP.Sample` object (*page 42*)

OnFingerGone(Object, String)

Fires when a user removes a finger from a fingerprint reader.

Syntax

```
public void OnFingerGone(
  Object Capture,
  String ReaderSerialNumber
);
```

Parameters

`Capture`	A `DPFP.Capture.Capture` object (*page 46*)
`ReaderSerialNumber`	If `Capture(Priority)` or `Capture()` was used to initialize an instance of the `Capture` class, this parameter is a **string** that contains an empty reader ID, that is, all zeros. If `Capture(String, Priority)` or `Capture(String)` was used to initialize an instance of the `Capture` class, this parameter is a **string** that contains the specified fingerprint reader serial number.

OnFingerTouch(Object, String)

Fires when a user touches a fingerprint reader.

Syntax

```
public void OnFingerTouch(
  Object Capture,
  String ReaderSerialNumber
);
```

Parameters

`Capture`	A `DPFP.Capture.Capture` object (*page 46*)
`ReaderSerialNumber`	If `Capture(Priority)` or `Capture()` was used to initialize an instance of the `Capture` class, this parameter is a **string** that contains an empty reader ID, that is, all zeros. If `Capture(String, Priority)` or `Capture(String)` was used to initialize an instance of the `Capture` class, this parameter is a **string** that contains the specified fingerprint reader serial number.

OnReaderConnect(Object, String)

Fires when a fingerprint reader is attached to a system.

Syntax

```
public void OnReaderConnect(
  Object Capture,
  String ReaderSerialNumber
);
```

Parameters

`Capture`	A `DPFP.Capture.Capture` object (*page 46*)
`ReaderSerialNumber`	**String** that contains the reader ID of the fingerprint reader from which the fingerprint sample was captured

OnReaderDisconnect(Object, String)

Fires when a fingerprint reader is disconnected from a system.

Syntax

```
public void OnReaderDisconnect(
  Object Capture,
  String ReaderSerialNumber
);
```

Parameters

`Capture`	A `DPFP.Capture.Capture` object (*page 46*)
`ReaderSerialNumber`	**String** that contains the reader ID of the fingerprint reader from which the fingerprint sample was captured

OnSampleQuality(Object, String, CaptureFeedback)

Fires when the quality of a fingerprint sample is verified.

Syntax

```
public void OnSampleQuality(
  Object Capture,
  String ReaderSerialNumber,
  CaptureFeedback CaptureFeedback
);
```

Parameters

`Capture`	A **`DPFP.Capture.Capture`** object (*page 46*)
`ReaderSerialNumber`	**String** that contains the reader ID of the fingerprint reader from which the fingerprint sample was captured
`CaptureFeedback`	Enumeration that specifies one of the values from **`DPFP.Capture.CaptureFeedback`** (*page 63*)

Libraries

DPFPShrNet.dll for the **`DPFP.Capture.CaptureFeedback`** enumeration

DPFPDevNet.dll for all other members of the **`DPFP.Capture`** namespace

GUI Component

The GUI component is wrapped within the **`DPFP.Gui`** namespace, which provides graphical user interfaces for performing fingerprint enrollment and fingerprint verification operations and an event handler status enumeration. The following two namespace are wrapped within the **`DPFP.Gui`** namespace:

- **`Enrollment`**—This namespace is defined in *Enrollment Namespace* on *page 70*.
- **`Verification`**—This namespace is defined in *Verification Namespace* on *page 80*.

Public Enumeration

EventHandlerStatus

Defines the codes that are returned by the fingerprint enrollment control and the fingerprint verification control event handlers to indicate the status of an operation.

Syntax

```
public enum EventHandlerStatus
{
  Success = 0,
  Failure = 1
};
```

Members

`Success`	An operation was performed successfully.
`Failure`	An operation failed.

Remarks

The members of this enumeration are used by the **`OnDelete`** and **`OnEnroll`** events (*page 75*) of the fingerprint enrollment control event handler and by the **`OnComplete`** event of the fingerprint verification control event handler (*page 82*).

Library

DPFPGuiNet.dll

Enrollment Namespace

The members of the **`DPFP.Gui.Enrollment`** namespace include a .NET control that implements a graphical user interface (described in *DPFP.Gui.Enrollment Graphical User Interface* on *page 94*) and provides the following functionality:

- Captures fingerprint samples from a fingerprint reader(s)
- Creates fingerprint feature sets for the purpose of enrollment
- Creates fingerprint templates
- Notifies an application when an enrollee commits to delete a fingerprint template
- Fires events

DPFP.Gui.Enrollment.EnrollmentControl Class

Provides a .NET control that is used for performing fingerprint enrollment operations.

Public Constructor

EnrollmentControl()

Initializes a new instance of the **`EnrollmentControl`** class that provides a .NET control for performing fingerprint enrollment operations.

Syntax

```
public EnrollmentControl()
```

Public Properties

EnrolledFingerMask

Returns or assigns the mask representing the user's enrolled fingerprints. The enrollment mask is 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.

Syntax

```
public int EnrolledFingerMask
```

Possible Values

`EnrolledFingerMask`	**Int** that receives or assigns the value of the fingerprint mask. All possible values are listed in *Table 4*.

Table 4. Values for the enrollment mask

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

This optional property is read/write. If you do not set it, the value `0` is used, which means that no fingerprints have been enrolled.

EventHandler

Loads a fingerprint enrollment control event handler. Set this property to `null` to clear all registered event handlers.

IMPORTANT: At least one event handler should be loaded to receive events.

Syntax

```
public EventHandler EventHandler
```

Possible Values

`EventHandler`	A `DPFP.Gui.Enrollment.EventHandler` object (*page 73*)

This property is write-only.

MaxEnrollFingerCount

Returns or assigns the value for the maximum number of fingerprints that can be enrolled.

Syntax

```
public int MaxEnrollFingerCount
```

Possible Values

`MaxEnrollFingerCount`	**Int** that receives or assigns the value for the maximum number of fingerprints that can be enrolled. Possible values are `1` through `10`.

This optional property is read/write. If you do not set it, the value `10` is used, which means the user can enroll all ten fingerprints.

ReaderSerialNumber

Returns or assigns the serial number of the fingerprint reader from which a fingerprint sample is captured.

Syntax

```
public String ReaderSerialNumber
```

Possible Values

`ReaderSerialNumber`	**String** that receives or assigns a fingerprint reader serial number

This optional property is read/write. If you do not set it, the following value is used: `{00000000-0000-0000-0000-000000000000}`. This means that the application will receive events from any of the fingerprint readers attached to the system.

DPFP.Gui.Enrollment.EventHandler Interface

Defines the fingerprint enrollment control events.

Syntax

```
public interface EventHandler
{
  void OnDelete(Object, int, ref Gui.EventHandlerStatus);
  void OnEnroll(Object, int, Template, ref Gui.EventHandlerStatus);
  void OnFingerTouch(Object, String, int);
  void OnFingerRemove(Object, String, int);
  void OnComplete(Object, String, int);
  void OnReaderConnect(Object, String, int);
  void OnReaderDisconnect(Object, String, int);
  void OnSampleQuality(Object, String, int, Capture.CaptureFeedback);
  void OnStartEnroll(Object, String, int);
  void OnCancelEnroll(Object, String, int);
}
```

Events

OnCancelEnroll(Object, String, int)

Fires when enrollment is canceled.

Syntax

```
public void OnCancelEnroll(
  Object Control,
  String ReaderSerialNumber,
  int Finger
);
```

Parameters

`Control`	A **`DPFP.Gui.Enrollment.EnrollmentControl`** object (*page 70*).
`ReaderSerialNumber`	Serial number of the fingerprint reader used for enrollment.
`Finger`	**Int** that contains the index value for the enrolled fingerprint. For possible values, see Table 5 (*page 74)*.

The **`Finger`** parameter is the index value of the finger associated with a fingerprint to be enrolled or 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 graphical user interface. All possible values are listed in *Table 5*.

Table 5. Finger index values in ANSI/NIST-ITL 1

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

OnComplete(Object, String, int)

Fires when a fingerprint sample is successfully captured by a fingerprint reader.

Syntax

```
public void OnComplete(
  Object Control,
  String ReaderSerialNumber,
  int Finger
);
```

Parameters

`Control`	A **`DPFP.Gui.Enrollment.EnrollmentControl`** object (*page 70*).
`ReaderSerialNumber`	Serial number of the fingerprint reader used for enrollment.
`Finger`	**Int** that contains the index value for the enrolled fingerprint. For possible values, see Table 5 (*page 74)*.

OnDelete(Object, int, ref Gui.EventHandlerStatus)

Fires when a user commits to delete an enrolled fingerprint. The application handles the deletion of the fingerprint template from a fingerprint data storage subsystem and can display its own success or error messages.

Syntax

```
public void OnDelete(
  Object Control,
  int Finger,
  ref Gui.EventHandlerStatus EventHandlerStatus
);
```

Parameters

Control	A **DPFP.Gui.Enrollment.EnrollmentControl** object (*page 70*)
Finger	**Int** that contains the index value of the (enrolled) fingerprint to be deleted. For possible values, see Table 5 (*page 74)*.
EventHandlerStatus	Enumeration that receives and contains one of the values from **DPFP.Gui.EventHandlerStatus**, which is set by the event handler, if needed (*page 69*).

OnEnroll(Object, int, Template, ref Gui.EventHandlerStatus)

Fires when a user enrolls a fingerprint, and returns a fingerprint template. The application handles the storage of the fingerprint template in a fingerprint data storage subsystem and can display its own success or error messages.

Syntax

```
public void OnEnroll(
  Object Control,
  int Finger,
  Template Template,
  ref Gui.EventHandlerStatus EventHandlerStatus
);
```

Parameters

`Control`	A **`DPFP.Gui.Enrollment.EnrollmentControl`** object (*page 70*)
`Finger`	**Int** that contains the index value for the enrolled fingerprint. For possible values, see Table 5 (*page 74)*.
`Template`	A **`DPFP.Template`** object (*page 43*)
`EventHandlerStatus`	Enumeration that receives and contains one of the values from **`DPFP.Gui.EventHandlerStatus`** (*page 69*)

OnFingerRemove(Object, String, int)

Fires when a user removes their finger from a fingerprint reader.

Syntax

```
public void OnFingerRemove(
  Object Control,
  String ReaderSerialNumber,
  int Finger
);
```

Parameters

`Control`	A **`DPFP.Gui.Enrollment.EnrollmentControl`** object (*page 70*).
`ReaderSerialNumber`	Serial number of the fingerprint reader used for enrollment.
`Finger`	**Int** that contains the index value for the enrolled fingerprint. For possible values, see Table 5 (*page 74)*.

OnFingerTouch(Object, String, int)

Fires when a user touches a fingerprint reader.

Syntax

```
public void OnFingerRemove(
  Object Control,
  String ReaderSerialNumber,
  int Finger
);
```

Parameters

`Control`	A `DPFP.Gui.Enrollment.EnrollmentControl` object (*page 70*).
`ReaderSerialNumber`	Serial number of the fingerprint reader used for enrollment.
`Finger`	**Int** that contains the index value for the enrolled fingerprint. For possible values, see Table 5 (*page 74)*.

OnReaderConnect(Object, String, int)

Fires when a fingerprint reader is connected.

Syntax

```
public void OnReaderConnect(
  Object Control,
  String ReaderSerialNumber,
  int Finger
);
```

Parameters

`Control`	A `DPFP.Gui.Enrollment.EnrollmentControl` object (*page 70*).
`ReaderSerialNumber`	Serial number of the fingerprint reader used for enrollment.
`Finger`	**Int** that contains the index value for the enrolled fingerprint. For possible values, see Table 5 (*page 74)*.

OnReaderDisconnect(Object, String, int)

Fires when a fingerprint reader is disconnected.

Syntax

```
public void OnReaderDisconnect(
  Object Control,
  String ReaderSerialNumber,
  int Finger
);
```

Parameters

`Control`	A `DPFP.Gui.Enrollment.EnrollmentControl` object (*page 70*).
`ReaderSerialNumber`	Serial number of the fingerprint reader used for enrollment.
`Finger`	**Int** that contains the index value for the enrolled fingerprint. For possible values, see Table 5 (*page 74)*.

OnSampleQuality(Object, String, int, Capture.CaptureFeedback)

Fires when the quality of a fingerprint reader sample is verified.

Syntax

```
public void OnSampleQuality(
  Object Control,
  String ReaderSerialNumber,
  int Finger,
  Capture.CaptureFeedback CaptureFeedback
);
```

Parameters

`Control`	A `DPFP.Gui.Enrollment.EnrollmentControl` object (*page 70*).
`ReaderSerialNumber`	Serial number of the fingerprint reader used for enrollment.
`Finger`	**Int** that contains the index value for the enrolled fingerprint. For possible values, see Table 5 (*page 74)*.
`CaptureFeedback`	Captured sample quality.

OnStartEnroll(Object, String, int)

Fires when enrollment has begun.

Syntax

```
public void OnStartEnroll(
  Object Control,
  String ReaderSerialNumber,
  int Finger
);
```

Parameters

`Control`	A **`DPFP.Gui.Enrollment.EnrollmentControl`** object (*page 70*).
`ReaderSerialNumber`	Serial number of the fingerprint reader used for enrollment.
`Finger`	**Int** that contains the index value for the enrolled fingerprint. For possible values, see Table 5 (*page 74)*.

Library

DPFPGuiNet.dll

Verification Namespace

The verification user control component is wrapped within the **`DPFP.Gui.Verification`** namespace. The members of this component include a .NET control that implements a graphical user interface (described in *DPFP.Gui.Verification Graphical User Interface* on *page 103*) and provides the following functionality:

- Receives fingerprint reader connect and disconnect event notifications
- Captures fingerprint samples from a fingerprint reader(s)
- Creates fingerprint feature sets for the purpose of verification
- Fires an event

DPFP.Gui.Verification.VerificationControl Class

Provides a .NET control that is used for performing fingerprint verification operations.

Public Constructor

VerificationControl()

Initializes a new instance of the **`VerificationControl`** class that provides a .NET control for performing fingerprint verification.

Syntax

```
public VerificationControl()
```

Public Properties

Active

Controls the fingerprint capture state.

Syntax

```
public bool Active
```

Possible Values

`True`	Allows fingerprint capture.
`False`	Disables fingerprint capture.

EventHandler

Loads a fingerprint verification control event handler. Set this property to **`null`** to clear all registered event handlers.

IMPORTANT: At least one event handler should be loaded to receive events.

Syntax

```
public EventHandler EventHandler
```

Possible Values

`EventHandler`	A **`DPFP.Gui.Verification.EventHandler`** object (*page 81*)

This property is write-only.

ReaderSerialNumber

Returns or assigns the serial number of the fingerprint reader from which a fingerprint sample is captured.

Syntax

```
public String ReaderSerialNumber
```

Possible Values

`ReaderSerialNumber`	**String** that receives or contains a fingerprint reader serial number

This optional property is read/write. If you do not set it, the following value is used: **`{00000000-0000-0000-0000-000000000000}`**. This means that the application will receive events from any of the fingerprint readers attached to the system.

DPFP.Gui.Verification.EventHandler Interface

Defines the fingerprint verification control events.

Syntax

```
public interface EventHandler
{
  void OnComplete(Object, FeatureSet, ref Gui.EventHandlerStatus);
}
```

Event

OnComplete(Object, FeatureSet, ref Gui.EventHandlerStatus)

Fires when a fingerprint feature set created for the purpose of verification is ready for comparison and returns the fingerprint feature set. The application handles the comparison of the fingerprint feature set with a fingerprint template.

Syntax

```
public void OnComplete(
  Object Control,
  FeatureSet FeatureSet,
  ref Gui.EventHandlerStatus EventHandlerStatus
);
```

Parameters

Control	A **DPFP.Gui.Verification.VerificationControl** object (*page 80*)
FeatureSet	A **DPFP.FeatureSet** object (*page 41*)
EventHandlerStatus	Enumeration that receives and contains one of the values from **DPFP.Gui.EventHandlerStatus**, which is set by the fingerprint verification control event handler, if needed (*page 69*)

Library

DPFPGuiNet.dll

Processing Component

The processing component is wrapped within the `DPFP.Processing` namespace. The members of this component provide methods and properties for

- Creating fingerprint feature sets for the purpose of enrollment or verification
- Performing the system function of fingerprint enrollment by creating fingerprint templates

DPFP.Processing.FeatureExtraction Class

Performs *fingerprint feature extraction*. The members of the `FeatureExtraction` class create a fingerprint feature set for the purpose of enrollment or verification by applying fingerprint feature extraction to a fingerprint sample.

Public Constructor

FeatureExtraction()

Initializes a new instance of the `FeatureExtraction` class for creating fingerprint feature sets.

Syntax

```
public FeatureExtraction()
```

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.
Internal	Failed to create context	The SDK failed to create a handle for processing, possibly due to misconfiguration, incompatible binaries, or security/memory limitations.

Public Method

CreateFeatureSet(Sample, DataPurpose, ref CaptureFeedback, ref FeatureSet)

Applies fingerprint feature extraction to a fingerprint sample and then creates a fingerprint feature set for the specified purpose.

Syntax

```
public void CreateFeatureSet(
  Sample Sample,
  DataPurpose Purpose,
  ref CaptureFeedback CaptureFeedback,
  ref FeatureSet FeatureSet
)
```

Parameters

`FingerprintSample`	A **`DPFP.Sample`** object (*page 42*)
`Purpose`	Enumeration that contains one of the values from **`DPFP.Processing.DataPurpose`** (*page 88*)
`CaptureFeedback`	Enumeration that receives and contains one of the values from **`DPFP.Capture.CaptureFeedback`** (*page 63*)
`FeatureSet`	A **`DPFP.FeatureSet`** object (*page 41*)

Exception

DPFP.Error.ErrorCodes	Message	Reason
InvalidFeatureSet	FeatureSet extraction failed	Fingerprint feature set extraction failed because the fingerprint sample is either corrupt or contains no features. For specifics, check the value of the **`CaptureFeedback`** parameter.
InvalidFeatureSetType	Invalid FeatureSet purpose	The fingerprint feature set purpose is incompatible.
InvalidParameter	Invalid Parameter	The arguments provided for this function are either null or contain no valid data.
Internal	FeatureSet extraction failed	Fingerprint feature set extraction failed, possibly due to memory or security limitations.

DPFP.Processing.Enrollment Class

Performs the system function of *fingerprint enrollment*. The members of the **`Enrollment`** class create a fingerprint template from a specified number of fingerprint feature sets that were created for the purpose of enrollment.

Public Constructor

Enrollment()

Initializes a new instance of the **`Enrollment`** class for performing the system function of fingerprint enrollment.

Syntax

```
public Enrollment()
```

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.
Internal	Failed to create context	The SDK failed to create a handle for processing, possibly due to misconfiguration, incompatible binaries, or security/memory limitations.

Public Methods

AddFeatures(FeatureSet)

Adds fingerprint feature sets, one-by-one, to a fingerprint template. The fingerprint template is complete when the **`TemplateStatus`** property is set to the value **`Ready`**.

Syntax

```
public void AddFeatures(
   FeatureSet FeatureSet
)
```

Parameter

`FeatureSet`	A **`DPFP.FeatureSet`** object (*page 41*)

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.
InvalidParameter	Invalid Parameter	The arguments provided for this function are either null or contain no valid data.
InvalidFeatureSet	Enrollment procedure failed	Fingerprint enrollment failed. The supplied fingerprint feature sets are insufficient or incompatible.
Internal	Enrollment procedure failed	Fingerprint enrollment failed, possibly due to memory or security limitations.

Clear()

Clears a fingerprint template and sets the value of the **`TemplateStatus`** property to **`Unknown`** so an application can begin another fingerprint template creation operation.

Syntax

```
public void Clear()
```

Public Properties

FeaturesNeeded

Returns the number of fingerprint feature sets still needed to create a fingerprint template. When the value of **`FeaturesNeeded`** is equal to `0`, the fingerprint template is created.

Syntax

```
public uint FeaturesNeeded
```

Possible Values

`FeaturesNeeded`	**Uint** that receives the value of the number of fingerprint feature sets

This property is read-only.

Template

Returns a **`DPFP.Template`** object created during a fingerprint enrollment operation.

IMPORTANT: The application should check the **`TemplateStatus`** property before reading this property.

Syntax

```
public Template Template
```

Possible Values

`Template`	A **`DPFP.Template`** object (*page 43*)

This property is read-only.

Exception

DPFP.Error.ErrorCodes	Message	Reason
InvalidParameter	Incomplete Enrollment	The fingerprint template is not ready to be retrieved. Check the **`TemplateStatus`** property before reading this property.

TemplateStatus

Returns a value that specifies the status of a fingerprint template creation operation.

Syntax

```
public Status TemplateStatus
```

Possible Values

`TemplateStatus`	Enumeration that receives one of the values from **`DPFP.Processing.Enrollment.Status`** (*page 88*)

This property is read-only.

Public Enumeration

Status

Defines the status of a fingerprint template creation operation.

Syntax

```
public enum Status
{
  Unknown = 0,
  Insufficient,
  Failed,
  Ready
}
```

Members

`Unknown`	The status of a template creation operation is not known, probably because a fingerprint template does not exist yet.
`Insufficient`	A fingerprint template exists, but more fingerprint feature sets are required to complete it.
`Failed`	A fingerprint template creation operation failed.
`Ready`	A fingerprint template was created and is ready for use.

Remarks

The members of this enumeration are used by the **`DPFP.Capture.TemplateStatus`** property (*page 87*).

DPFP.Processing.DataPurpose Enumeration

Defines the purpose for which a fingerprint feature set is to be used.

Syntax

```
public enum DataPurpose
{
  Unknown = 0,
  Verification,
  Enrollment
};
```

Members

`Unknown`	The purpose is not known.
`Verification`	A fingerprint feature set to be used for the purpose of verification.
`Enrollment`	A fingerprint feature set to be used for the purpose of enrollment.

Remarks

The members of this enumeration are used by the **`DPFP.Processing.FeatureExtraction.CreateFeatureSet`** method (*page 83*).

Libraries

DPFPShrNet.dll for the **`DPFP.Processing.DataPurpose`** enumeration

DPFPEngNet.dll for all other members of the **`DPFP.Processing`** namespace

Verification Component

The verification component is wrapped within the `DPFP.Verification` namespace. The members of this component provide a method and properties for

- Performing the system function of fingerprint verification
- Returning and retrieving the false accept rate (FAR)
- Returning the results of the fingerprint verification operation

DPFP.Verification.Verification Class

Performs the system function of *fingerprint verification*, which is a one-to-one comparison of a fingerprint feature set with a fingerprint template produced at enrollment that returns a decision of match or non-match.

Public Constructors

Verification()

Initializes a new instance of the `Verification` class for comparing a fingerprint feature set with a fingerprint template using the default value of the false accept rate (FAR). (For more information about the FAR, see *False Positives and False Negatives* on *page 20*.)

Syntax

```
public Verification()
```

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.
Internal	Failed to create context	The SDK failed to create a handle for processing, possibly due to misconfiguration, incompatible binaries, or security/memory limitations.

Verification(int)

Initializes a new instance of the `Verification` class for comparing a fingerprint feature set with a fingerprint template and assigns the value of the FAR. (For more information about the FAR, see *False Positives and False Negatives* on *page 20*.)

IMPORTANT: Although the default value 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 111* for more information about setting the value of the FAR.

Syntax

```
public Verification(int FARRequested)
```

Parameter

`FARRequested`	**Int** that contains the value of the requested FAR

Exception

DPFP.Error.ErrorCodes	Message	Reason
NotInitialized	Failed to initialize	The SDK failed to initialize properly, possibly due to misconfiguration, incompatible binaries, or security/ memory limitations.
Internal	Failed to create context	The SDK failed to create a handle for processing, possibly due to misconfiguration, incompatible binaries, or security/memory limitations.

Method

Verify(FeatureSet, Template, ref Result)

Performs the system function of fingerprint verification and specifies a comparison decision based on the FAR set by the **`FARRequested`** property.

Syntax

```
public void Verify(
  FeatureSet FeatureSet,
  Template Template,
  ref Result Result
)
```

Parameters

`FeatureSet`	A **`DPFP.FeatureSet`** object, where the **`Purpose`** parameter of the **`DPFP.Processing.FeatureExtraction.CreateFeatureSet`** method was set to the value **`Verification`** (*page 83*)
`Template`	A **`DPFP.Template`** object (*page 43*)
`Result`	A **`DPFP.Verification.Result`** object (*page 93*)

Exception

DPFP.Error.ErrorCodes	Message	Reason
InvalidParameter	Invalid Parameter	The arguments provided for this function are either null or contain no valid data.
InvalidFeatureSet	Verification Failure	Fingerprint verification failed. The supplied fingerprint feature set or foundering template are insufficient or incompatible.
Internal	Verification Failure	Fingerprint verification failed, possibly due to memory/ security limitations.

Properties

FARRequested

Returns or assigns the requested false accept rate (FAR). (For more information about the FAR, see *False Positives and False Negatives* on *page 20*.)

IMPORTANT: Although the default value 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 111* for more information about setting the value of the FAR.

Syntax

```
public int FARRequested
```

Possible Values

`FARRequested`	**Int** that receives or assigns the value of the requested FAR

This optional property is read/write. You can use the **`FARRequested`** property to check or modify the value of the FAR before calling the **`Verify`** method. If you do not set this property, the default value is used.

DPFP.Verification.Result Class

Represents the results of a fingerprint verification operation.

Default Constructor

The `DPFP.Verification.Result` object is supplied in the `Result` parameter of the `Verify` method (*page 91*). Default construction sets the `FARAchieved` property to `0` and the `Verified` property to false.

Properties

FARAchieved

Returns or assigns the value of the achieved FAR for a comparison operation.

Syntax

```
public int FARAchieved
```

Possible Values

`FARAchieved`	**Int** that receives or assigns the value of the FAR that was achieved for the comparison

This property is read/write. See *Achieved FAR* on *page 113* for more information about this property.

Verified

Returns or assigns the comparison decision, which indicates whether the comparison of a fingerprint feature set and a fingerprint template resulted in a decision of match or non-match. This decision is based on the value of the `FARRequested` property (*page 92*).

Syntax

```
public bool Verified
```

Possible Values

`Verified`	**Boolean** that receives or assigns the comparison decision. Possible values are true for a decision of match or false for a decision of non-match.

This property is read/write.

Library

DPFPVerNet.dll

Graphical User Interfaces 6

This chapter describes the functionality of the graphical user interfaces that are wrapped within the following namespaces:

- **`DPFP.Gui.Enrollment`**

  This namespace includes the graphical user interface described in the next section. The constructor, properties, and event handler contained within this namespace are described on *page 70*.

- **`DPFP.Gui.Verification`**

  This object includes the graphical user interface described on *page 103*. The constructor, properties, and event handler contained within this namespace are described on *page 80*.

DPFP.Gui.Enrollment Graphical User Interface

The graphical user interface included with the **`DPFP.Gui.Enrollment.EnrollmentControl`** object consists of two elements. The first element is used to provide instructions for selecting a fingerprint to enroll or to unenroll (delete) and is used to indicate already-enrolled fingerprints. The second element is used to provide instructions and feedback, both graphically and textually, about the enrollment process.

The tables and figure in this section describe the interaction between the user and the graphical user interface during fingerprint enrollment and unenrollment (deletion).

NOTE: In the tables, the elements are referred to as the *hands element* and the *numbers element*.

Enrolling a Fingerprint

Figure 9 illustrates the fingerprint enrollment process using the **`DPFP.Gui.Enrollment.EnrollmentControl`** object graphical user interface. Picture numbers in the figure correspond to the pictures in Table 6 on *page 96*. *Table 6* illustrates and describes the interaction between the user and the graphical user interface during fingerprint enrollment.

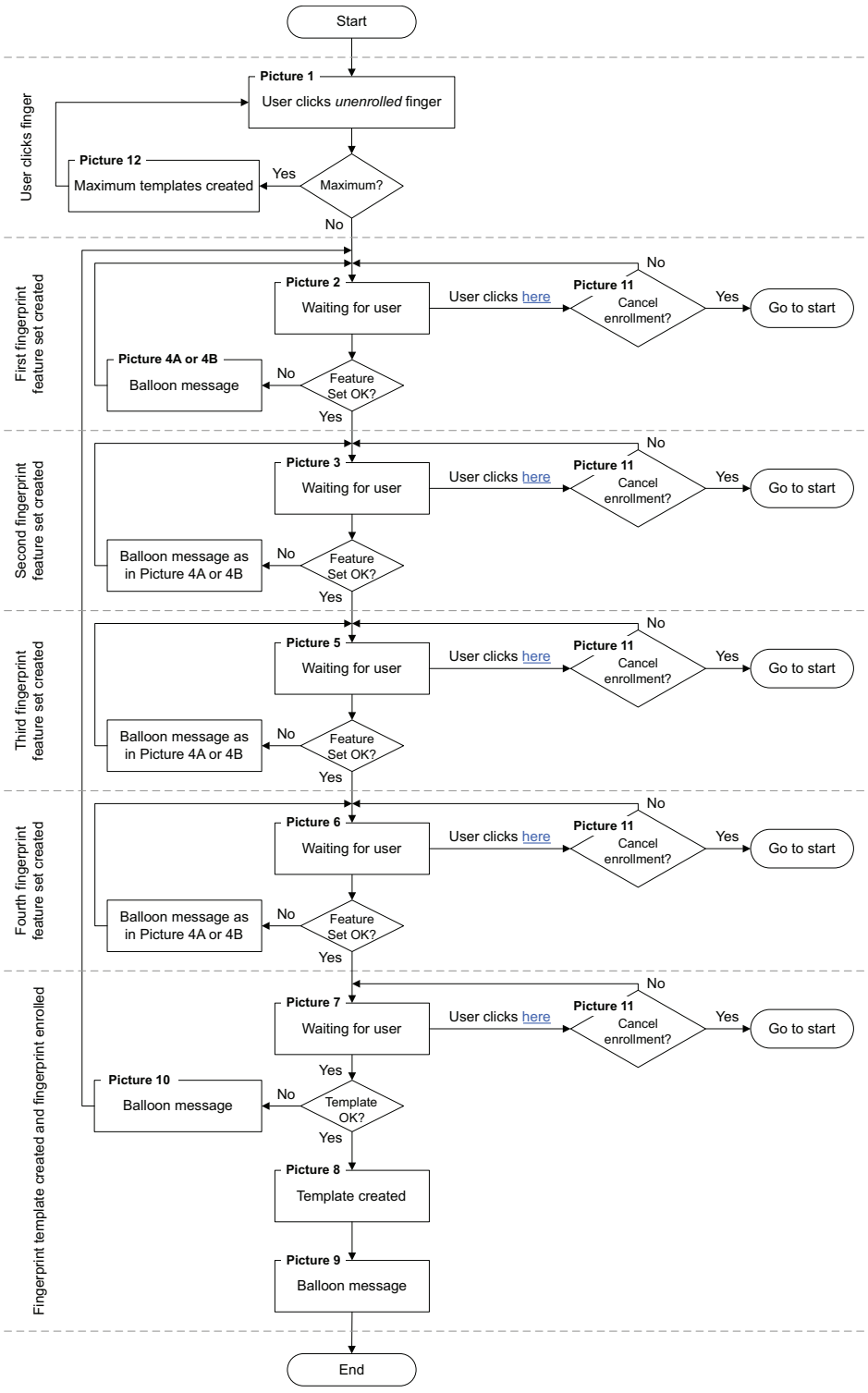


Figure 9. Enrolling a fingerprint using the **`DPFP.Gui.Enrollment.EnrollmentControl`** object graphical user interface

Table 6. `DPFP.Gui.Enrollment.EnrollmentControl` object graphical user interface: Enrolling a fingerprint

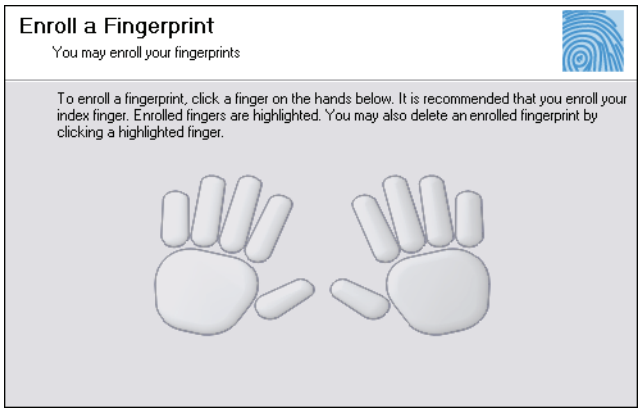
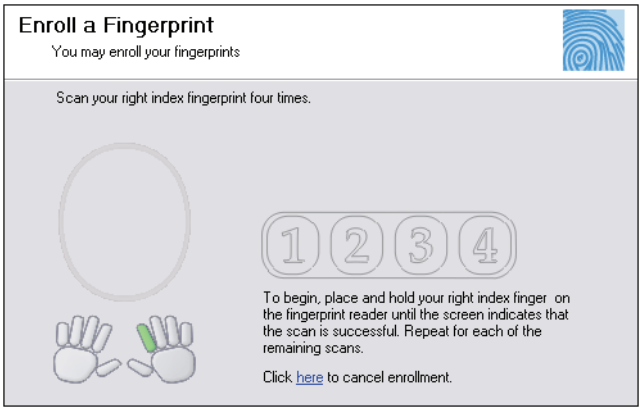
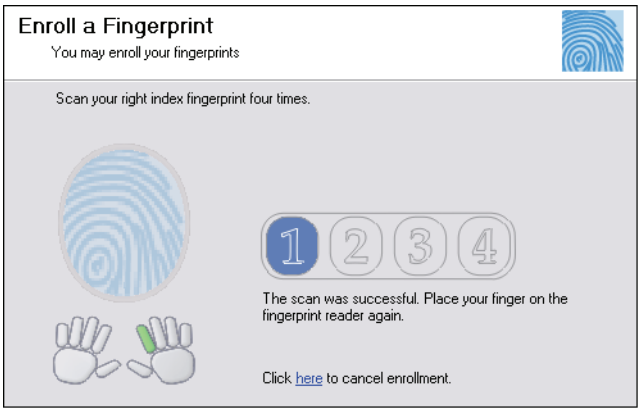
Graphical user interface	User actions and user interface feedback
Picture 1 	This image indicates that no fingerprints have been enrolled, because the fingers associated with any enrolled fingerprints are green.
Picture 2 	The user clicked the right index finger, and control was passed from the hands element to the numbers element. The numbers element is ready to enroll the user's right index fingerprint, as indicated by the green finger on the hand in the bottom left corner.
Picture 3 	The user touched the fingerprint reader, and a fingerprint feature set was created.

Table 6. `DPFP.Gui.Enrollment.EnrollmentControl` object graphical user interface: Enrolling a fingerprint *(continued)*

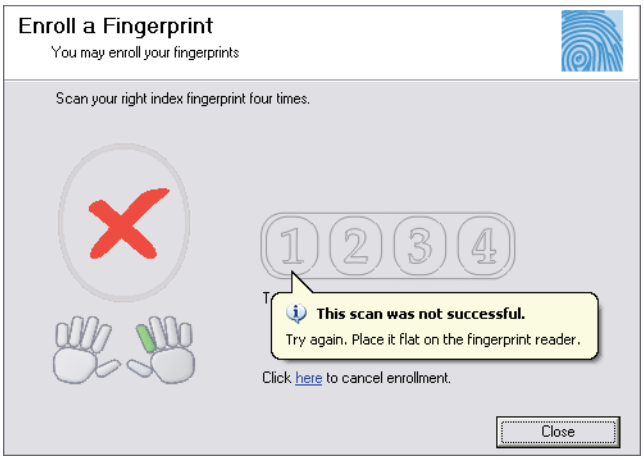
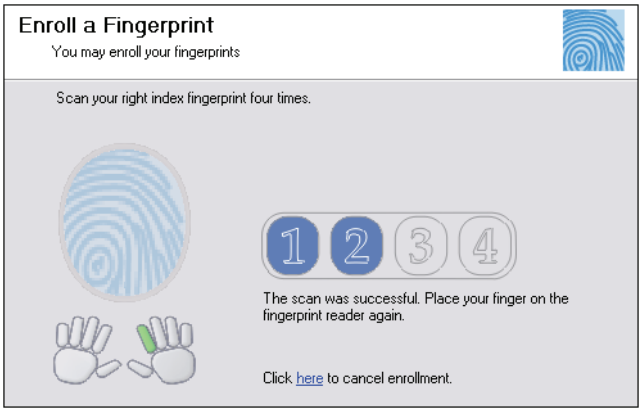
Graphical user interface	User actions and user interface feedback
Picture 4A  Picture 4B  	The user touched the fingerprint reader, but a fingerprint feature set was not created. The message that is displayed depends on the quality of the fingerprint sample, as shown in Pictures 4A and 4B.
Picture 5  	The user touched the fingerprint reader, and a second fingerprint feature set was created.

Table 6. `DPFP.Gui.Enrollment.EnrollmentControl` object graphical user interface: Enrolling a fingerprint *(continued)*

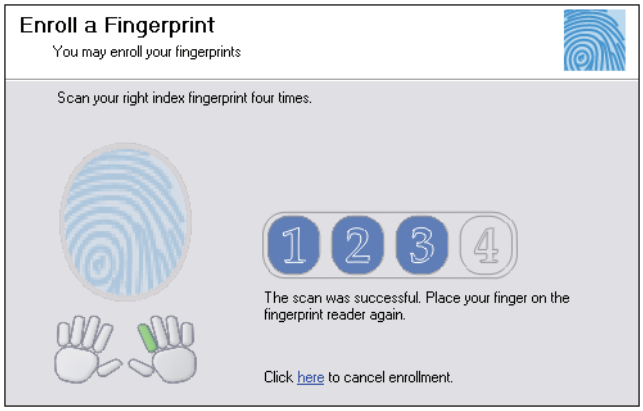
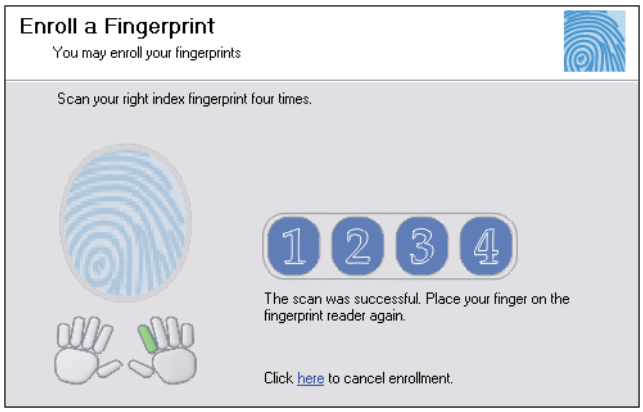
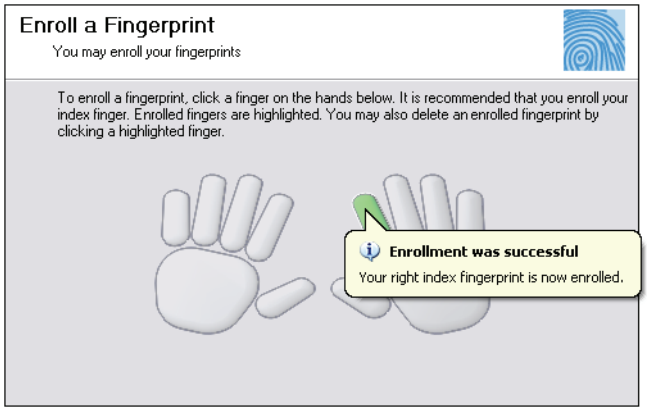
Graphical user interface	User actions and user interface feedback
Picture 6 	The user touched the fingerprint reader, and a third fingerprint feature set was created.
Picture 7 	The user touched the fingerprint reader, and a fourth fingerprint feature set was created.
Picture 8 	When a fingerprint template is created for the selected finger, control is passed to the hands element. This image appears when the `OnEnroll` event of the enrollment control event handler is fired and returns a status of `Success` in the `EventHandlerStatus` parameter.

Table 6. `DPFP.Gui.Enrollment.EnrollmentControl` object graphical user interface: Enrolling a fingerprint *(continued)*

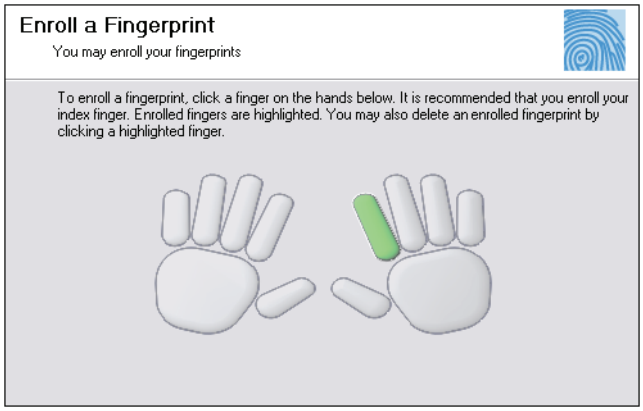
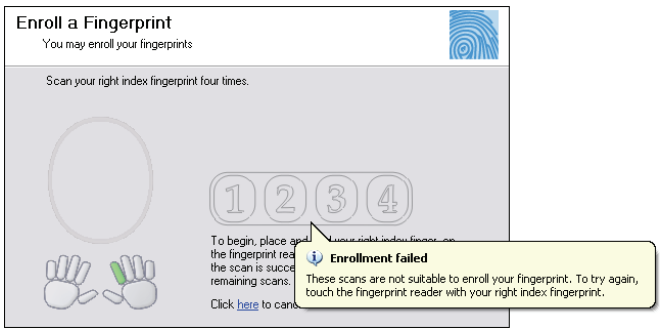
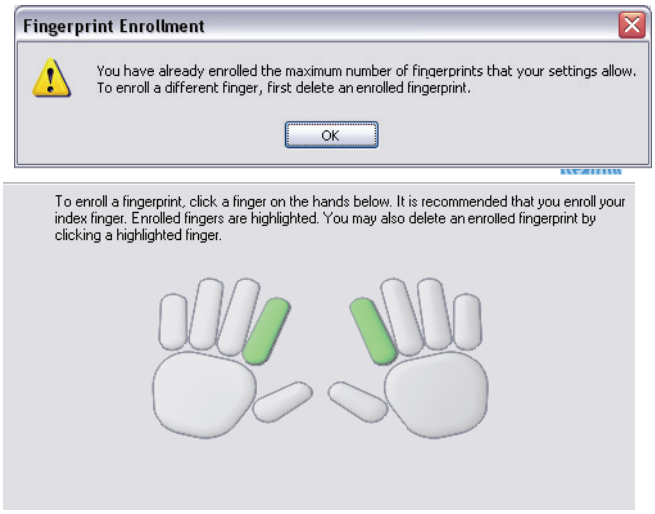
Graphical user interface	User actions and user interface feedback
Picture 9 	The hands element indicates that the right index fingerprint is enrolled, that is, the finger is green. The value of the `EnrolledFingerMask` property is `000000000 001000000`, or 64.
Picture 10 	A fingerprint template was not created for the selected finger. The user is instructed to try again, and control remains with the numbers element.
Picture 11 	This message appears when the user clicks **here** in **Click here to cancel enrollment**. When the user clicks **No**, this message is dismissed and control is returned to the numbers element. When the user clicks **Yes**, this message is dismissed and control is passed to the hands element. The user can cancel enrollment at any time by clicking **here** and then clicking **Yes**.

Table 6. `DPFP.Gui.Enrollment.EnrollmentControl` object graphical user interface: Enrolling a fingerprint *(continued)*

Graphical user interface	User actions and user interface feedback
Picture 12 	This message is displayed when a user who has already enrolled the maximum allowed number of fingerprints (set by the `MaxEnrollFingerCount` property) clicks a finger associated with an unenrolled finger in the hands element. When the user clicks **OK**, control is returned to the hands element.

Unenrolling (Deleting) a Fingerprint

The table below illustrates and describes the interaction between the user and the graphical user interface during fingerprint unenrollment (deletion).

Table 7. `DPFP.Gui.Enrollment.EnrollmentControl` graphical user interface: Unenrolling (deleting) a fingerprint template

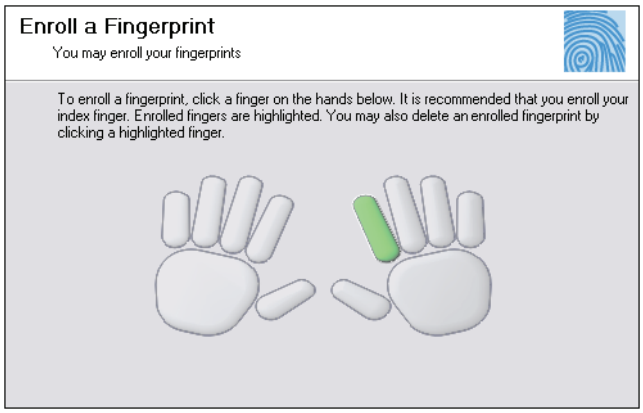
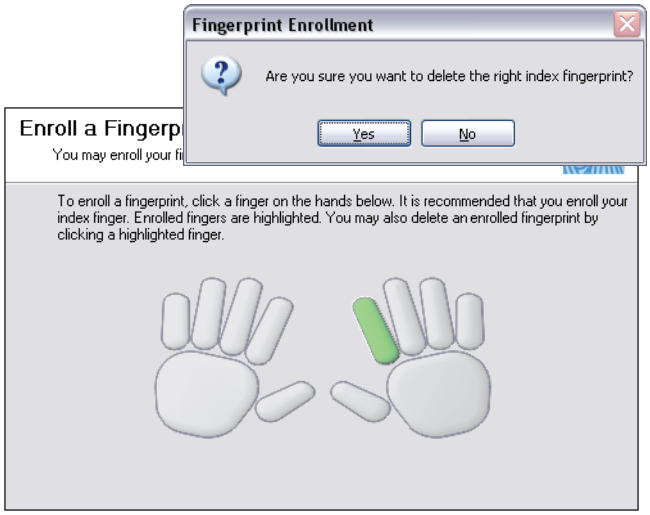
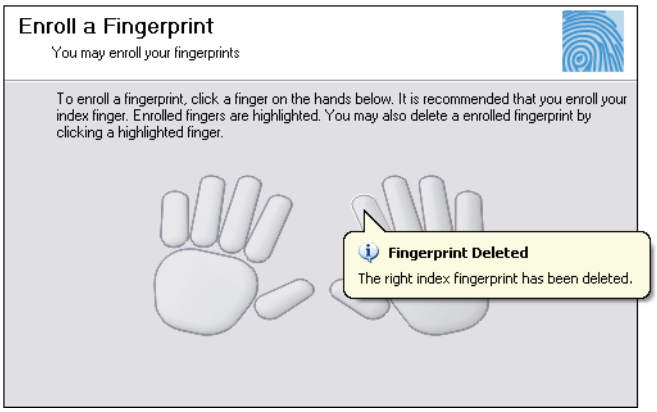
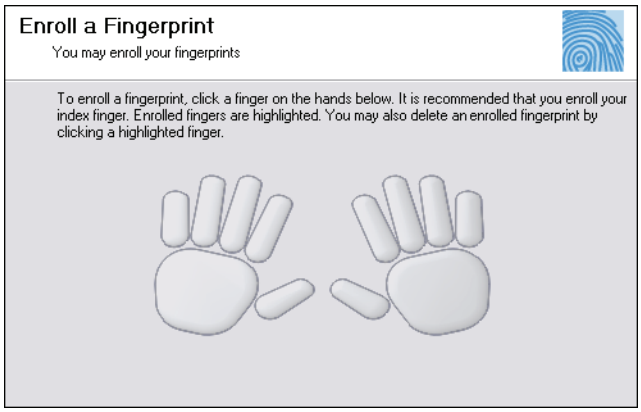
Graphical user interface	User actions and user interface feedback
	The hands element indicates that the right index fingerprint is enrolled, that is, the finger is green. The value of the `EnrolledFingerMask` property is `000000000 001000000`, or 64.
	This message appears when the user clicks the right index fingerprint (which was previously enrolled). When the user clicks **No**, this message is dismissed and control is returned to the hands element, which remains unchanged. When the user clicks **Yes**, this message is dismissed and control is returned to the hands element, where the **Fingerprint Deleted** message is displayed (see the next picture).

Table 7. `DPFP.Gui.Enrollment.EnrollmentControl` graphical user interface: Unenrolling (deleting) a fingerprint template *(continued)*

Graphical user interface	User actions and user interface feedback
	This image appears when the `OnDelete` event of the enrollment control event handler is fired and returns a status of `Success` in the `EventHandlerStatus` parameter. When an application receives this event, it should delete the fingerprint template associated with the right index finger. The value of the `EnrolledFingerMask` property is now set to `000000000 000000000`, or 0.
	The green color is removed from the right index finger, indicating that the associated fingerprint is no longer enrolled.

DPFP.Gui.Verification Graphical User Interface

The graphical user interface included with the `DPFP.Gui.Verification.VerificationControl` object consists of one element. This element is used to indicate the connection status of the fingerprint reader and to provide feedback about the fingerprint verification process. *Table 8* illustrates and describes the interaction between the user and the graphical user interface.

Table 8. `DPFP.Gui.Verification` graphical user interface

Graphical user interface	User actions and user interface feedback
	Indicates that the fingerprint reader is connected and ready for the user to scan a finger.
	Indicates that the fingerprint reader is disconnected.
	Indicates a comparison decision of match from a fingerprint verification operation. This image appears when the `OnComplete` event of the verification control event handler is fired and returns a status of `Success` in the `EventHandlerStatus` parameter.
	Indicates a comparison decision of non-match from a fingerprint verification operation. This image appears when the `OnComplete` event of the verification control event handler is fired and returns a status of `Failure` in the `EventHandlerStatus` parameter.
	Indicates that the fingerprint sample capture operation failed.

Developing Citrix-aware applications 7

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.dll, 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 within the product 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 within the DigitalPersona Pro Server installation package.

Redistribution 8

You may redistribute the files in the RTE\Install and the Redist folders in the One Touch for Windows SDK software package 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

The following table indicates which merge modules are required to support each development language and OS.

Merge module	C/C++		COM/ActiveX		.NET		Java	
	32-bit	64-bit	32-bit	64-bit	32-bit	64-bit	32-bit	64-bit
DpDrivers.msm	X	X	X	X	X	X	X	X
DpPolicies_OTW.msm	X	X	X	X	X	X	X	X
DpCore.msm	X	X	X	X	X	X	X	X
DpCore_x64.msm		X		X		X		X
DpProCore.msm	X		X		X		X	
DpProCore_x64.msm		X		X		X		X

Merge module	C/C++		COM/ActiveX		.NET		Java	
DpFpRec.msm	X		X		X		X	
DpFpRec_x64.msm		X		X		X		X
DpFpUI.msm	X	X	X	X	X	X	X	X
DpFpUI_x64.msm		X		X		X		X
DpOTCOMActX.msm			X	X	X	X		
DpOTCOMActX_x64.msm				X		X		
DpOTDotNet.msm					X	X		
DpOTShrDotNet.msm					X	X		
DpOTJni.msm							X	X
DpOTJni_x64.msm								X
DpOTJava.msm							X	X

The merge modules, and the files that they contain, are listed below alphabetically.

- 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 files:

  - <*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:
  - DPFPDevNET.dll
  - DPFPEngNET.dll
  - DPFPVerNET.dll
  - DPFPGuiNET.dll
  - DPFPCtlXTypeLibNET.dll
  - DPFPCtlXWrapperNET.dll
  - DPFPShrXTypeLibNET.dll
- DpOTShrDotNET.msm

  This merge module contains the following files:
  - DPFPShrNET.dll

Fingerprint Reader Documentation

You may redistribute the documentation included in the Redist folder in the 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 in the 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 *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

The DigitalPersona Fingerprint Recognition Engine supports the representation for the FAR probability that fully conforms to the BIOAPI 1.1, BioAPI 2.0, and UPOS standard specifications. In this representation, the probability is represented as a positive 32-bit integer, or zero. (Negative values are reserved for special uses.)

The definition PROBABILITY_ONE provides a convenient way of using this representation. PROBABILITY_ONE has the value 0x7FFFFFFF (where the prefix 0x denotes base 16 notation), which is 2147483647 in decimal notation. If the probability (P) is encoded by the value (INT_N), then

$$INT\_N = P * PROBABILITY\_ONE$$

$$P = \frac{INT\_N}{PROBABILITY\_ONE}$$

Probability P 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.

The resultant value of INT_N is represented in column two, in decimal notation.

Table 2. Common values of probability and resultant INT_N values

Probability (P)	Value of INT_N in decimal notation
0.001 = 0.1% = 1/1000	2147483
0.0001 = 0.01% = 1/10000	214748
0.00001 = 0.001% = 1/100000	21475
0.000001 = 0.0001% = 1/1000000	2147

Requested FAR

You specify the value of the FAR, which is INT_N from the previous equation, using the **`FARRequested`** property (*page 92*). While you can request any value from 0 to the value PROBABILITY_ONE, 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.

Specifying the FAR in C#

If you are developing your application in C#, you specify the value of the FAR (INT_N) in the **`FARRequested`** property of the **`DPFP.Verification.Verification`** object. The following sample code sets the FAR to a value of 0.000001, or 0.0001%.

```
static int PROBABILITY_ONE = 0x7FFFFFFF;

DPFP.Verification.Verification verification = new
   DPFP.Verification.Verification();

...

//Sets the FAR to 0.0001%
verification.FARRequested = PROBABILITY_ONE / 1000000;
```

You can also set the value of the FAR through construction, as shown in the following sample code.

```
static int PROBABILITY_ONE = 0x7FFFFFFF;

//Constructs and sets the FAR to 0.0001%
DPFP.Verification.Verification verification =
   new DPFP.Verification.Verification(PROBABILITY_ONE / 1000000);
```

Specifying the FAR in Visual Basic

If you are developing your application in Visual Basic, you specify the value of the FAR (INT_N) in the **`FARRequested`** property of the **`DPFP.Verification.Verification`** object. The following sample code sets the FAR to a value of 0.0001, or 0.01%.

```
Const PROBABILITY_ONE as Long = &H7FFFFFFF

Dim verification as new DPFP.Verification.Verification()

...

' Sets the FAR to 0.01%
verification.FARRequested = PROBABILITY_ONE / 10000
```

You can also set the value of the FAR through construction, as shown in the following sample code.

```
Const PROBABILITY_ONE as Long = &H7FFFFFFF

' //Constructs and sets the FAR to 0.01%
Dim verification as new DPFP.Verification.Verification(PROBABILITY_ONE / 10000)
```

Achieved FAR

The actual value of the FAR achieved for a particular verification operation is returned in the **`DPFP.Verification.Result.FARAchieved`** property (*page 93*). 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 enrollment templates.

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

```
#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++

```
    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
```

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.

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 object

An object that inherits the properties of a DPFP.Data object. Fingerprint data objects include DPFP.Sample (represents a fingerprint sample), DPFP.FeatureSet (represents a fingerprint feature set), and DPFP.Template (represents a fingerprint template).

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 sample.

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), intermediate (after some processing), or processed.

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

E

F

S

T

U

V

W