DigitalPersona, Inc.

One Touch® for Windows SDK

Java Edition

Version 1.6

Developer Guide

digitalPersona.

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: Java Edition is a software development tool for integrating fingerprint biometrics into a wide set of Java-based applications, services, and products. It 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 and an existing fingerprint template.

Other editions of the One Touch for Windows SDK enable developers to use a variety of programming languages in a number of development environments (Visual Basic, C++ and .NET) to create their applications. Each edition 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: Java Edition 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: Java Edition 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

Note that to take advantage of this feature, your fingerprint-enabled application must run on the Terminal Services or Citrix server, not on the client. If you are developing a Citrix-aware application, see additional information in Chapter 8, *Developing Citrix-aware applications,* on *page 100.*

Target Audience

This guide is for developers who have a working knowledge of the Java programming language.

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: Java Edition, identifies the minimum system requirements needed to run the SDK, and lists the DigitalPersona products and fingerprint templates supported by the SDK.

Chapter 2, *Quick Start*, provides a quick introduction to the One Touch for Windows SDK: Java 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: Java Edition terminology and concepts. This chapter also includes typical workflow diagrams and explanations of the One Touch for Windows SDK: Java 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 SDK: Java Edition API.

Chapter 6, *Graphical User Interfaces*, describes the functionality of the graphical user interfaces included with the DPFPEnrollmentControl and DPFPVerificationControl objects.

Chapter 7, *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 SDK: Java Edition API.

Chapter 9, *Setting the False Accept Rate*, provides information about determining and using specific values for the FAR and evaluating and testing achieved values.

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 **`DPFPCapture. 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)

Additional Resources

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

- x86-based processor or better
- JRE or JDK 1.5 or 1.6 (needed to run samples and completed applications)
- USB connector on the computer where the fingerprint reader is to be connected

Supported DigitalPersona Hardware Products

The One Touch for Windows SDK: Java 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 the One Touch for Windows SDK are also compatible with the following DigitalPersona SDKs:

- Gold SDK
- Gold CE SDK
- One Touch for Linux SDK, all distributions

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

Quick Start 2

This chapter provides a quick introduction to the One Touch for Windows SDK: Java Edition using the included sample applications.

There are three sample applications.

- Fingerprint Enrollment & Verification Sample - Provides a basic UI for exploring fingerprint enrollment and verification and the events that are produced in the process. Also provides the ability to save and open a fingerprint template.
- Java UI Sample - Includes a professionally designed series of integrated dialogs that can be used for enrolling and unenrolling fingerprints.
- Java Console Sample - Creates a simple console-based application illustrating enrollment and verification, reader selection and adding a user to the fingerprint database.

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 19*.

Installation

Before you can use the sample applications, you must install the One Touch for Windows SDK: Java Edition, which includes the DigitalPersona One Touch for Windows Runtime Environment (RTE).

The Java runtime (JRE) or Java SDK (JDK) is required in order to run the sample applications, but is not required prior to installing the One Touch for Windows SDK: Java Edition.

To install the One Touch for Windows SDK: Java Edition

1. In the SDK\Install folder of the software package, launch the Setup.exe file, and then click **Next**. If installing on a 64-bit computer, use the setup.exe located in the SDK\Install\x64 folder.

2. Follow the installation instructions as they appear.
3. Restart your computer.
4. Optionally install a supported JRE or JDK. See page *4* for supported JRE/JDKs.

Connect the Fingerprint Reader

Insert the fingerprint reader into the USB connector on the system where you installed the SDK.

Using the Sample Applications

After installation, you will find the sample applications in the following folder,

<installation path>\One Touch SDK\Java\Samples

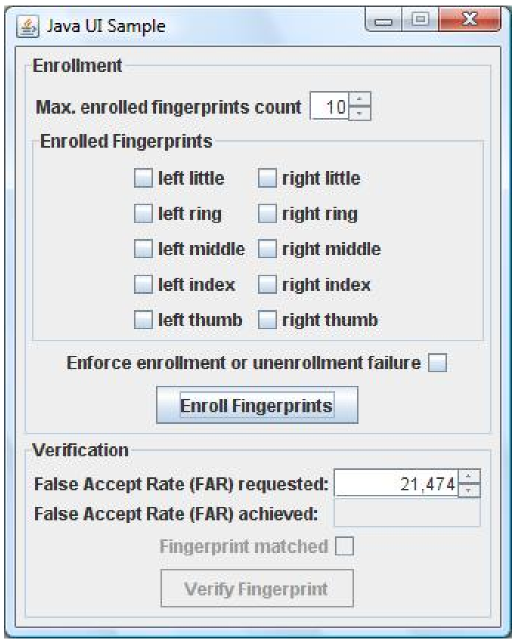
The sample applications are:

Name	**File to Launch**	**Page**
Java UI Sample	\Samples\uisupport\run.bat	*7*
Fingerprint Enrollment and Verification Sample	\Samples\enrollment\run.bat	*11*
Console UI Sample	\Samples\console\run.bat	*13*

Java UI Sample

To start the Java UI sample application -

1. Launch the run.bat file. in the Samples\uisupport folder.
2. The **Java UI Sample** dialog box displays.



Max. enrolled fingerprints count 10

Enforce enrollment or unenrollment failure

False Accept Rate (FAR) requested: 21,474

False Accept Rate (FAR) achieved:

Functions

This sample application illustrates the following enrollment functions:
Max. Enrolled Fingerprints Count - allows you to specify the maximum number of fingerprints that can be enrolled.
Enrolled Fingerprints - displays a checkmark next to each enrolled finger. Click on any enrolled finger to unenroll (delete) the finger.

Enforce enrollment or unenrollment failure - when checked, the enrollment and unenrollment processes will always fail.

Enroll Fingerprints - Click this button to enroll or unenroll (delete) a finger

This sample application illustrates the following verification functions:

False Accept Rate (FAR) requested - Illustrates setting the 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 Accept Rate (FAR) achieved - Illustrates the FAR actually achieved during the current transaction.

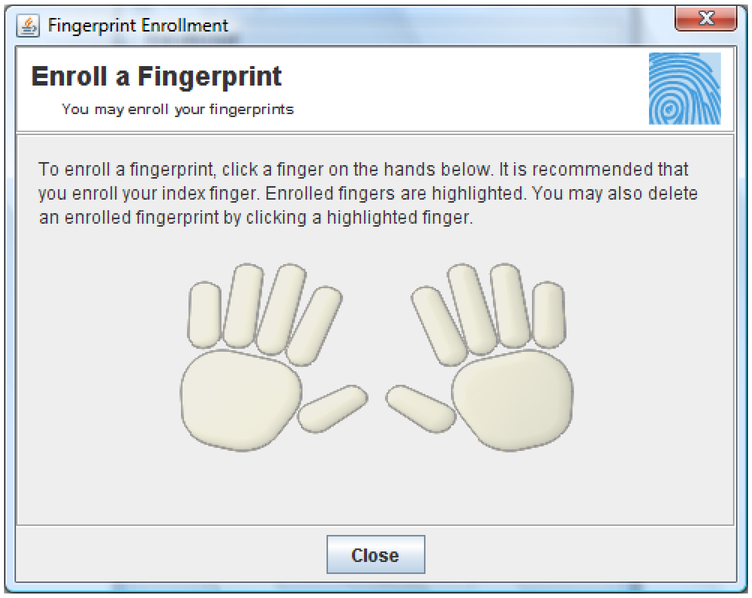
Fingerprint Matched - When verifying a fingerprint, if the scanned fingerprint matches a previously enrolled fingerprint, this box will be checked.

Verify Fingerprint - Click this button to verify a fingerprint. The button is disabled until at least one fingerprint has been enrolled.

Enrolling Fingerprints

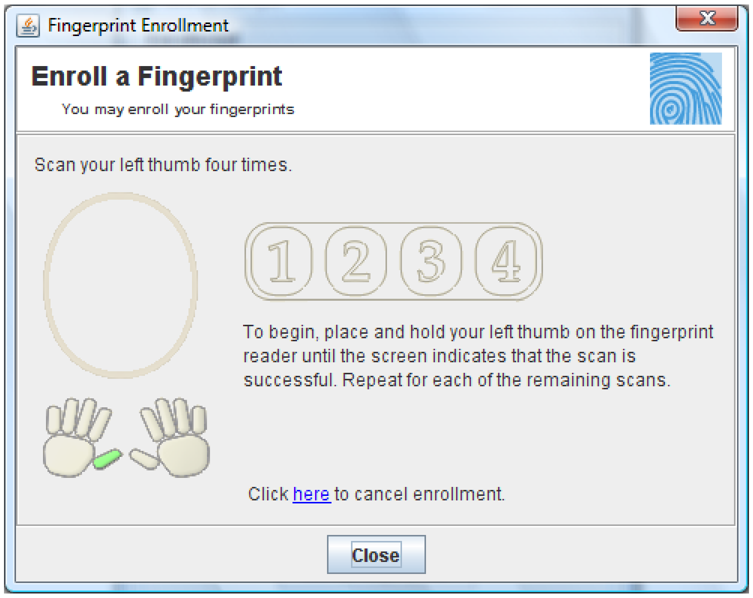
To enroll a fingerprint -

1. Click **Enroll Fingerprints**. The Fingerprint Enrollment dialog displays.



To enroll a fingerprint, click a finger on the hands below. It is recommended that you enroll your index finger. Enrolled fingers are highlighted. You may also delete an enrolled fingerprint by clicking a highlighted finger.

2. Click the finger on the illustration that corresponds to the finger that you wish to enroll.


Fingerprint Enrollment

Enroll a Fingerprint

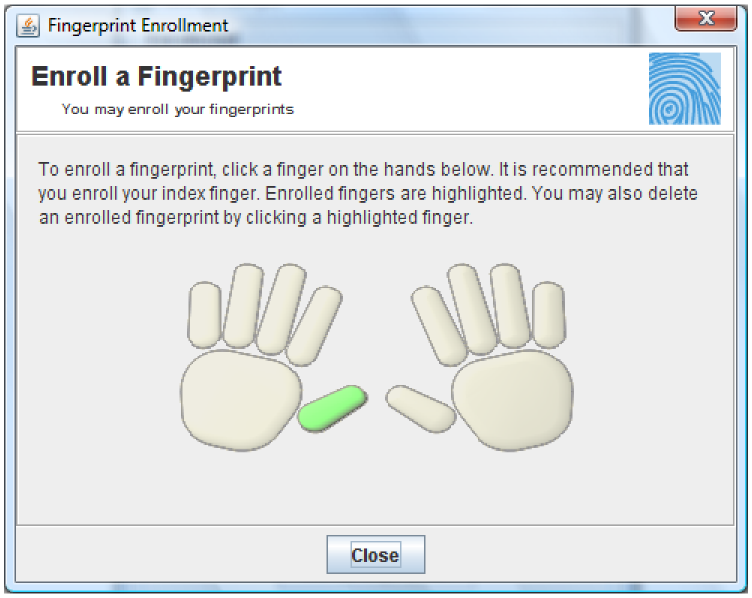
You may enroll your fingerprints

Scan your left thumb four times.

To begin, place and hold your left thumb on the fingerprint reader until the screen indicates that the scan is successful. Repeat for each of the remaining scans.

Click here to cancel enrollment.

3. Scan your fingerprint successfully four times. The previous screen will display, with the successfully enrolled finger highlighted in green.


Fingerprint Enrollment

Enroll a Fingerprint

You may enroll your fingerprints

To enroll a fingerprint, click a finger on the hands below. It is recommended that you enroll your index finger. Enrolled fingers are highlighted. You may also delete an enrolled fingerprint by clicking a highlighted finger.

Deleting an Enrolled Fingerprint

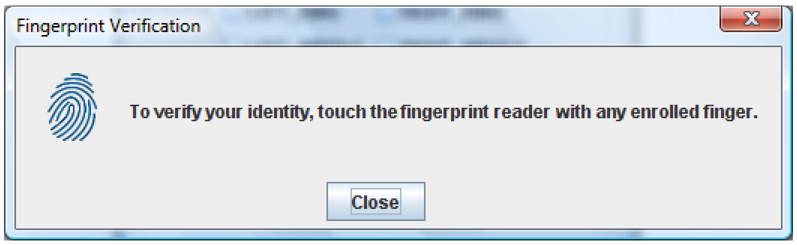
To delete an enrolled fingerprint -

- On the Java UI Sample dialog, click any enrolled finger, i.e. one with a checkmark in the box next to it.
- Or, on the Enroll a Fingerprint dialog, click any enrolled fingerprint, i.e. one that is highlighted in green.

Verifying a fingerprint

To verify a fingerprint -

1. Click **Verify Fingerprint**.


Fingerprint Verification

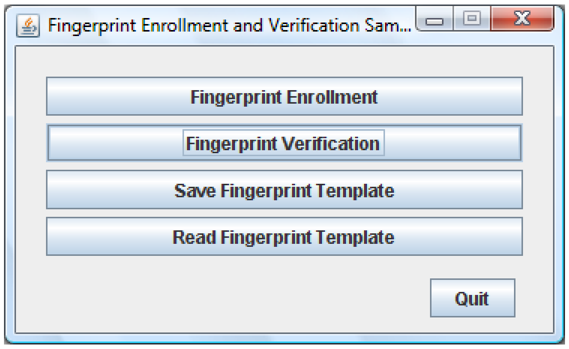
To verify your identity, touch the fingerprint reader with any enrolled finger.

2. Touch the fingerprint reader with any enrolled finger.

Fingerprint Enrollment and Verification Sample

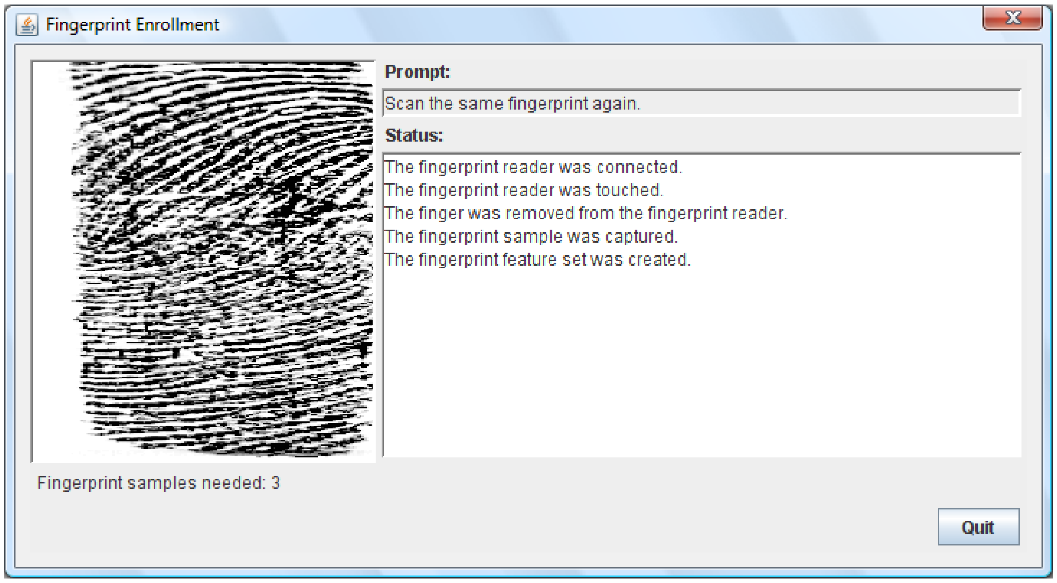
To start the Fingerprint Enrollment and Verification Sample application -

1. Launch the run.bat file in the Samples\Enrollment folder.
2. The **Fingerprint Enrollment and Verification Sample** dialog box displays.


Fingerprint Enrollment and Verification Sam...
Fingerprint Enrollment
Fingerprint Verification
Save Fingerprint Template
Read Fingerprint Template
Quit

Functions

This sample application illustrates the following functions:
Fingerprint Enrollment - illustrates default prompts and standard events occurring in the enrollment process. Also shows the image captured by the fingerprint reader.

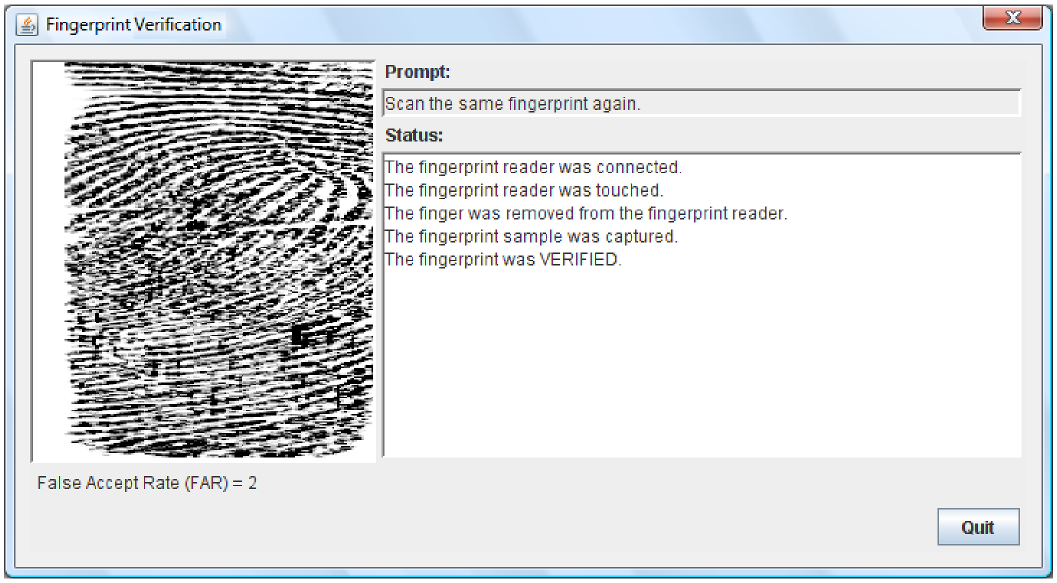


Scan the same fingerprint again.

```
The fingerprint reader was connected.
The fingerprint reader was touched.
The finger was removed from the fingerprint reader.
The fingerprint sample was captured.
The fingerprint feature set was created.
```

Fingerprint samples needed: 3

Fingerprint Verification - illustrates default prompts and standard events occurring in the verification process. Also shows image captured by the fingerprint reader.



```
Scan the same fingerprint again.
```

```
The fingerprint reader was connected.
The fingerprint reader was touched.
The finger was removed from the fingerprint reader.
The fingerprint sample was captured.
The fingerprint was VERIFIED.
```

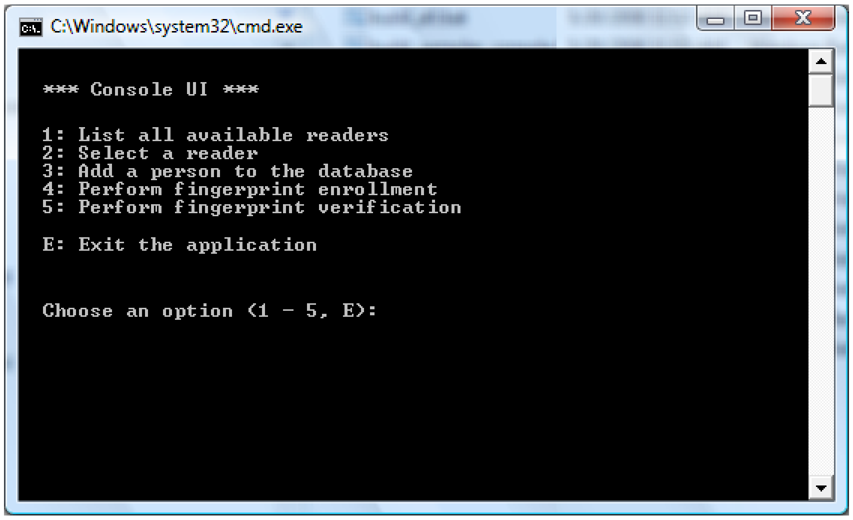
False Accept Rate (FAR) = 2

Save Fingerprint Template - illustrates saving the fingerprint template as a file using a standard Java Save dialog.
Read Fingerprint Template - illustrates opening and reading the fingerprint template using a standard Java Save dialog.

Console UI Sample

To start the Console UI Sample application -

1. Launch the run.bat file in the Samples\console folder.
2. The **Console UI** displays.



```
*** Console UI ***

1: List all available readers
2: Select a reader
3: Add a person to the database
4: Perform fingerprint enrollment
5: Perform fingerprint verification

E: Exit the application


Choose an option (1 - 5, E):
```

Functions

This sample application illustrates the following functions:
List all available readers - illustrates listing all available fingerprint readers.
Select a reader - illustrates selection of a specific fingerprint reader or an option to use any available reader.
Add a person to the database - illustrates adding a person to the user database. You must select a reader and add a person to the database before you can enroll their fingerprints.
Perform fingerprint enrollment - illustrates a console-based enrollment process. You must select a reader and add a person to the database before you can enroll a fingerprint.
Perform fingerprint verification - illustrates a console-based verification process. You must select a reader and add a person to the database and enroll a fingerprint before you can perform verification.
Exit the application - Closes the sample application and the command box.

Installation 3

This chapter contains instructions for installing the various components of the One Touch for Windows SDK: Java 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: Java 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: Java 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: Java 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 16* for the 32-bit installation and in Table 3 on *page 17* for the 64-bit installation, are also installed on your hard disk.

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

Folder	**Files/Description**
<installation folder>One Touch SDK\Java\Docs	DigitalPersona One Touch for Windows SDK Java Edition Developer Guide
<installation folder>One Touch SDK\Java\Samples	This folder contains subfolders for each of the samples provided as part of the SDK, with source code, build.bat and run.bat files in the directory. To run the sample applications use the run.bat file in the appropriate directory.
<installation folder>One Touch SDK\Java\Samples\UI Support	This folder contains the source code, build.bat, and run.bat files for the sample Java application, UISupport, that demonstrates the functionality of the graphical user interface.
<installation folder>One Touch SDK\Java\Samples\Enrollment	This folder contains the source code, build.bat, and run.bat files for the sample Java application, Enrollment, that shows how to use the One Touch for Windows SDK: Java Edition API for performing fingerprint enrollment and fingerprint verification.
<installation folder>One Touch SDK\Java\Samples\Console	This folder contains the source code, build.bat, and run.bat files for the sample Java application, console.bat, that demonstrates through a console application, enrollment and verification and additional functionality such as selecting a reader and adding a user to the user database.

Installing the Runtime Environment (RTE)

When you develop a product based on the One Touch for Windows SDK: Java 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 94* for licensing terms.)

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

To install the One Touch for Windows SDK: Java Edition RTE for 32-bit operating systems

1. In the RTE folder in the SDK software package, open the Setup.exe file.
2. Follow the installation instructions as they appear.

Table 2 identifies the files that are installed on your hard disk for 32-bit versions of the supported operating systems.

Table 2. One Touch for Windows SDK: Java Edition RTE installed files and folders, 32-bit installation

Folder	File	Description
<installation folder>\Bin	DPCOper2.dll DPDevice2.dll DPDevTS.dll DpHostW.exe DPmsg.dll DPMux.dll DpSvInfo2.dll DPTSClnt.dll	DLLs and executable file used by all of the One Touch for Windows APIs
<installation folder>\Bin\Java	dpfpenrollment.jar dpfpverification.jar dpotapi.jar dpotjni.jar	Java library files and executables used to create One Touch for Windows functionality.
<system folder>	DPFPApi.dll DpClback.dll dpHFtrEx.dll dpHMatch.dll DPFpUI.dll	DLLs used by all of the One Touch for Windows APIs
<system folder>	otdpfpjni.dll otfxjni.dll otmcjni.dll	DLLs used by the One Touch for Windows SDK: Java Edition API

To install the One Touch for Windows SDK: Java Edition RTE for 64-bit operating systems

1. In the RTE\x64 folder in the SDK software package, open the Setup.exe file.
2. Follow the installation instructions as they appear.

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

Table 3. One Touch for Windows SDK: Java Edition RTE installed files and folders, 64-bit installation

Folder	File	Description
<drive>\Program Files (x86)\Bin	DPCOper2.dll DPDevice2.dll DPDevTS.dll DpHostW.exe DPmsg.dll DPMux.dll DpSvInfo2.dll DPCrStor.dll	DLLs and executable file used by all of the One Touch for Windows APIs
<installation folder>\Bin	DPTSClnt.dll	64-bit DLLs used by all of the One Touch for Windows APIs
<installation folder>\Bin\Java	dpfpenrollment.jar dpfpverification.jar dpotapi.jar dpotjni.jar	Java library files and executables used to create One Touch for Windows functionality.

Table 3. One Touch for Windows SDK: Java 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 otdpfpjni.dll otfxjni.dll otmcjni.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 otdpfpjni.dll otfxjni.dll otmcjni.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: Java Edition 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: Java Edition concepts and terminology. (For more details on the subject of fingerprint biometrics, refer to the “DigitalPersona White Paper: Guide to Fingerprint Recognition” included in the One Touch for Windows SDK: Java Edition software package.) This chapter also includes typical workflow diagrams and explanations of the One Touch for Windows SDK: Java Edition API functions used to perform the tasks in the workflows.

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: Java 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 21*. Some of the tasks in these processes are done by the *fingerprint reader* and its driver; some are accomplished using One Touch for Windows SDK: Java 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: Java 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: Java Edition.)

1. *Obtain the Subject Identifier of the person to be verified.
2. Capture a fingerprint sample using the fingerprint reader.
3. Extract a fingerprint feature set for the purpose of verification from the fingerprint sample.
4. *Retrieve the fingerprint template associated with the Subject Identifier from your repository.

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

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

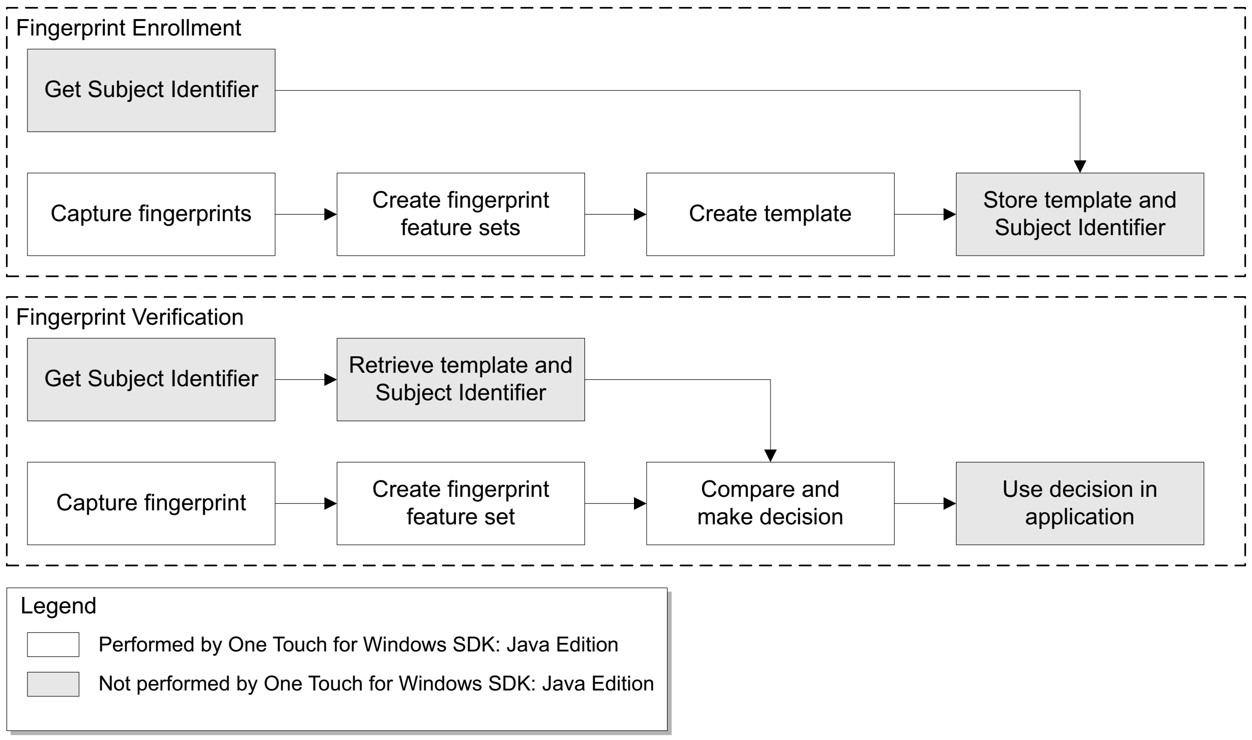

Fingerprint Enrollment
Get Subject Identifier
Capture fingerprints
Create fingerprint feature sets
Create template
Store template and Subject Identifier
Fingerprint Verification
Get Subject Identifier
Retrieve template and Subject Identifier
Capture fingerprint
Create fingerprint feature set
Compare and make decision
Use decision in application
Legend
Performed by One Touch for Windows SDK: Java Edition
Not performed by One Touch for Windows SDK: Java Edition

Figure 1. DigitalPersona fingerprint recognition system

Creating an object

In the workflows and their descriptions on the following pages, there are instructions such as “Create Capture object.” As you can see from the description, step 1 on *page 25*, this is done using the DPFPCapture class.

However, unlike in the .NET language, we cannot just call DPFPCapture capture = new DPFPCapture(), because DPFPCapture is an abstract interface, not a concrete class. To create a concrete class you must invoke some factory which will construct the object with the desired interface. The One Touch for Windows SDK: Java Edition provides standard factories via the DPFPGlobal object, so the construction code will be similar to -

DPFPCapture capture = DPFPGlobal.getCaptureFactory().createCapture();

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 SDK: Java 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 in the following workflows which are preceded by two asterisks (**) are performed by a fingerprint reader, while steps preceded by a single 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 SDK: Java 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.

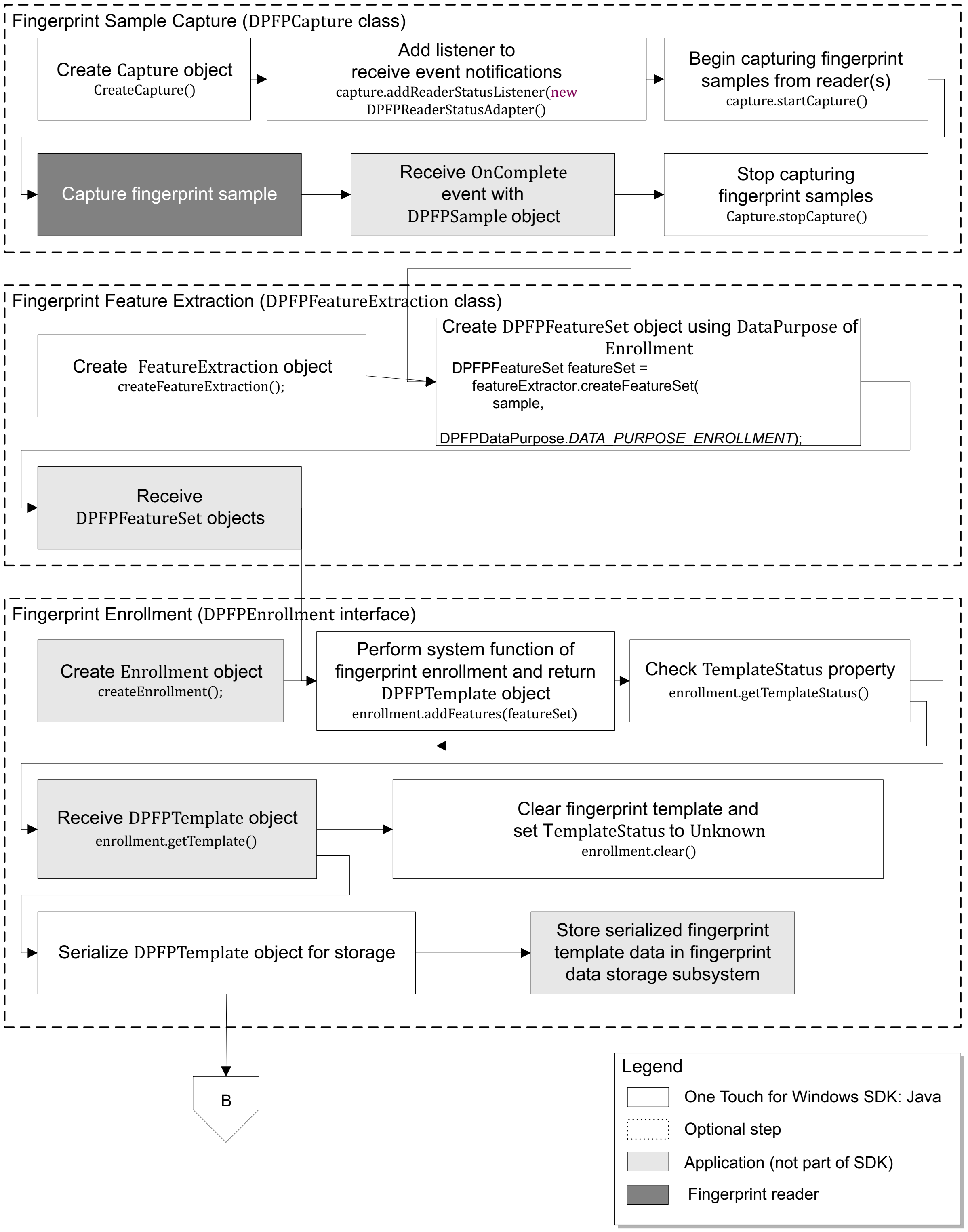

Fingerprint Sample Capture (DPFPCapture class)
Create Capture object
CreateCapture()
Add listener to receive event notifications
capture.addReaderStatusListener(new DPFPReaderStatusAdapter()
Begin capturing fingerprint samples from reader(s)
capture.startCapture()
Capture fingerprint sample
Receive OnComplete event with DPFPSample object
Stop capturing fingerprint samples
Capture.stopCapture()
Fingerprint Feature Extraction (DPFPFeatureExtraction class)
Create FeatureExtraction object
createFeatureExtraction();
Create DPFPFeatureSet object using DataPurpose of Enrollment
DPFPFeatureSet featureSet = featureExtractor.createFeatureSet(sample, DPFPDataPurpose.DATA_PURPOSE_ENROLLMENT);
Receive DPFPFeatureSet objects
Fingerprint Enrollment (DPFPEnrollment interface)
Create Enrollment object
createEnrollment();
Perform system function of fingerprint enrollment and return DPFPTemplate object
enrollment.addFeatures(featureSet)
Check TemplateStatus property
enrollment.getTemplateStatus()
Receive DPFPTemplate object
enrollment.getTemplate()
Clear fingerprint template and set TemplateStatus to Unknown
enrollment.clear()
Serialize DPFPTemplate object for storage
Store serialized fingerprint template data in fingerprint data storage subsystem
B
Legend
One Touch for Windows SDK: Java
Optional step
Application (not part of SDK)
Fingerprint reader

Figure 2. Typical fingerprint enrollment workflow

Fingerprint Sample Capture (DPFPCapture Class)

1. Create a new instance of the **`DPFPCapture`** class (*page 50*).
2. Load a fingerprint sample capture operation event handler for receiving event notifications by adding event listeners (*page 50*).
3. Begin capturing fingerprint samples from the fingerprint reader(s) connected to a system by calling the **`startCapture()`** method (*page 52*).
4. **Capture a fingerprint sample from a fingerprint reader. Note that you cannot change the priority or the reader(s) setting of a **`DPFPCapture`** object after it is started.
5. *Receive the **`dataAcquired()`** event from the fingerprint sample capture event handler along with a **`DPFPSample`** object when the fingerprint sample is successfully captured by the fingerprint reader (*page 54*).
6. *Pass the **`DPFPFeatureExtraction.createFeatureSet(DPFPSample sample, DPFPDataPurpose purpose);`** method. (See step 2 in the next section.)
7. Stop capturing fingerprint samples by calling the **`stopCapture`** method (*page 52*).

Fingerprint Feature Extraction (DPFPFeatureExtraction Class)

1. Create a new instance of the **`DPFPFeatureExtraction`** class (*page 65*).
2. Create **`DPFPFeatureSet`** objects by calling the **`createFeatureSet(DPFPSample sample, DPFPDataPurpose purpose)`** method using the value **`Enrollment`** for **`DataPurpose`** and passing the **`DPFPSample`** object from step 6 of the previous section (*page 65*).
3. *Pass the **`DPFPFeatureSet`** objects created in the previous step to the **`addFeatures`** method. (See step 2 in the next section.)

Fingerprint Enrollment (DPFPEnrollment Class)

1. Create a new instance of the **`DPFPEnrollment`** class (*page 64*).
2. Perform the system function of fingerprint enrollment by calling the **`addFeatures(featureSet)`** method and passing the **`DPFPFeatureSet`** objects from step 3 of the previous section (*page 64*).
3. Check the **`TemplateStatus`** property after each call to the **`addFeatures`** method using **`getTemplateStatus()`** "getter" method (*page 64*).

   When the **`TemplateStatus`** property returns the value `DPFPTemplateStatus.TEMPLATE_STATUS_READY`, a **`DPFPTemplate`** object is created.
4. *Receive the **`DPFPTemplate`** object.
5. Serialize the **`DPFPTemplate`** object (see *Serializing a Fingerprint Data Object* on *page 36*).

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 **`DPFPTemplateStatus.TEMPLATE_STATUS_UNKNOWN`** by calling the **`clear()`** method (*page 64*).

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 SDK: Java Edition API functions used to perform the tasks in the workflows.

Enrolling a Fingerprint

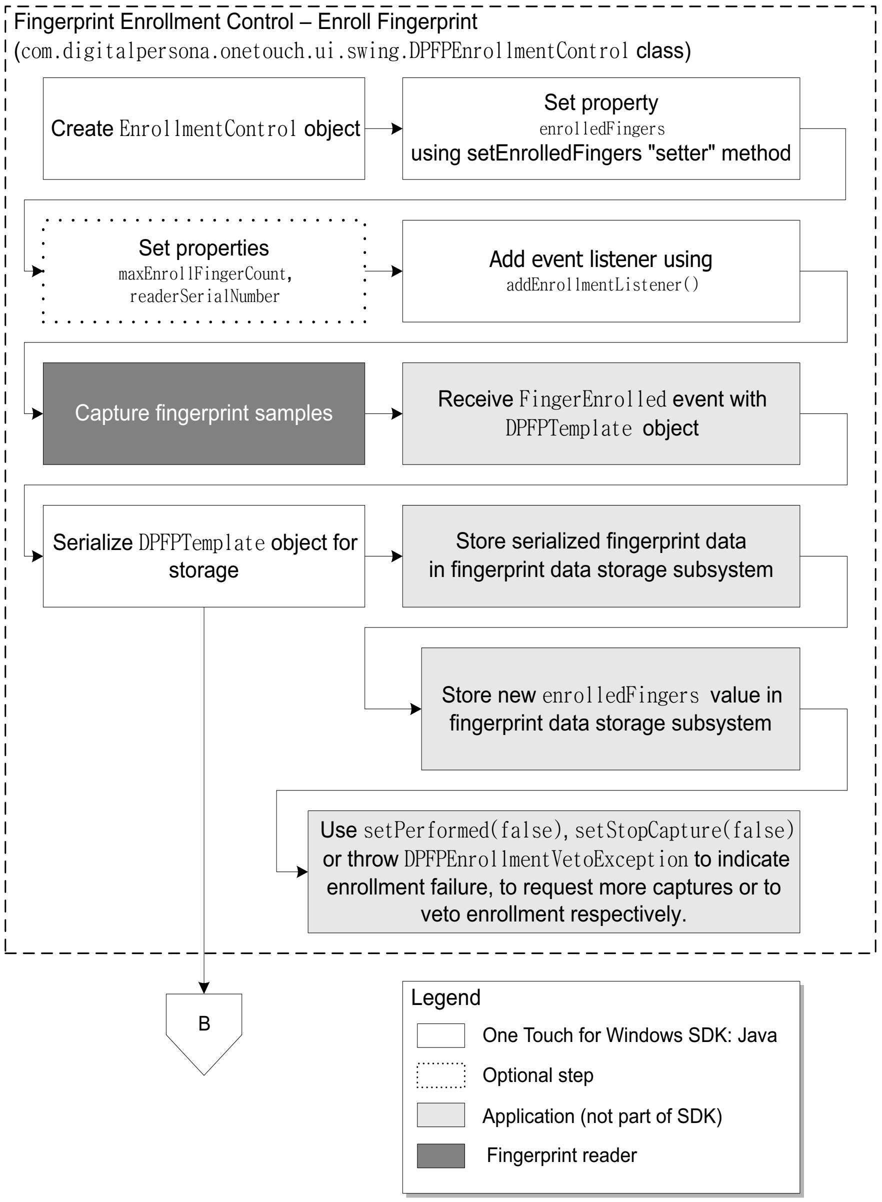

Fingerprint Enrollment Control – Enroll Fingerprint
(com.digitalpersona.onetouch.ui.swing.DPFPEnrollmentControl class)
Create EnrollmentControl object
Set property enrolledFingers using setEnrolledFingers "setter" method
Set properties maxEnrollFingerCount, readerSerialNumber
Add event listener using addEnrollmentListener()
Capture fingerprint samples
Receive FingerEnrolled event with DPFPTemplate object
Serialize DPFPTemplate object for storage
Store serialized fingerprint data in fingerprint data storage subsystem
Store new enrolledFingers value in fingerprint data storage subsystem
Use setPerformed(false), setStopCapture(false) or throw DPFPEnrollmentVetoException to indicate enrollment failure, to request more captures or to veto enrollment respectively.
B
Legend
One Touch for Windows SDK: Java
Optional step
Application (not part of SDK)
Fingerprint reader

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

1. Create a new instance of the **`com.digitalpersona.onetouch.ui.swing.DPFPEnrollmentControl`** class (*page 71*).
2. Set the **`enrolledFingers`** property **using the `setEnrolledFingers "setter" method`** (*page 72*).
3. Optionally, set the **`EnrollFingerCount`** and **`readerSerialNumber`** properties, using the **`setMaxEnrollFingerCount`** and **`setReaderSerialNumber`** methods (*page 72*).
4. Add an event listener for receiving event notifications using **`addEnrollmentListener()`** (*page 71*).
5. **Capture a predetermined number of fingerprint samples from a fingerprint reader.
6. *Receive the **`fingerEnrolled(DPFPEnrollmentEvent)`** event from the fingerprint enrollment control event handler, along with the **`DPFPTemplate`** object (*page 80*).
7. Serialize the **`DPFPTemplate`** object (see *Serializing a Fingerprint Data Object* on *page 36*).
8. *Store the serialized fingerprint template data and the new value of **`enrolledFingers`**, using **`getEnrolledFingers() getter`**, in a fingerprint data storage subsystem.
9. *Use the **`setPerformed(false)`** or **`setStopCapture(false)`** methods of the **`DPFPEnrollmentEvent;`** or throw **`DPFPEnrollmentVetoException`** to indicate enrollment failure, to request more captures or to veto enrollment (*page 73*).

Unenrolling (Deleting) a Fingerprint Template

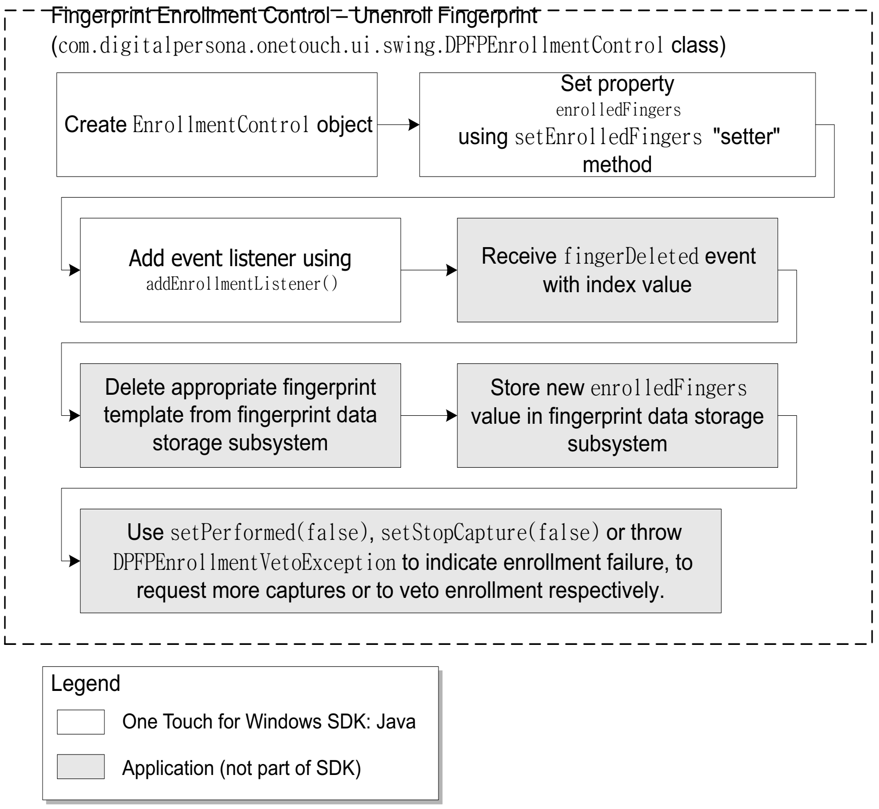

Fingerprint Enrollment Control – Unenroll Fingerprint
(com.digitalpersona.onetouch.ui.swing.DPFPEnrollmentControl class)
Create EnrollmentControl object
Set property enrolledFingers using setEnrolledFingers "setter" method
Add event listener using addEnrollmentListener()
Receive fingerDeleted event with index value
Delete appropriate fingerprint template from fingerprint data storage subsystem
Store new enrolledFingers value in fingerprint data storage subsystem
Use setPerformed(false), setStopCapture(false) or throw DPFPEnrollmentVetoException to indicate enrollment failure, to request more captures or to veto enrollment respectively.
Legend
One Touch for Windows SDK: Java
Application (not part of SDK)

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

1. Create a new instance of the **`com.digitalpersona.onetouch.ui.swing.DPFPEnrollmentControll`** class (*page 71*).
2. *Retrieve the value of the **`enrolledFingers`** property stored in the fingerprint data storage subsystem.
3. Set the **`enrolledFingers`** property **using the `setEnrolledFingers "setter" method`** (*page 72*).
4. Add an event listener for receiving event notifications using **`addEnrollmentListener()`** (*page 71*).
5. *Receive the **`fingerDeleted`** event from the enrollment control event handler, along with the finger index value (*page 80*).
6. *Delete the appropriate fingerprint template from the fingerprint data storage subsystem.
7. *Store the new value of **`enrolledFingers`** in the fingerprint data storage subsystem.

8. *Use the **`setPerformed(false)`**or **`setStopCapture(false)`** methods of the **`DPFPEnrollmentEvent;`** or throw **`DPFPEnrollmentVetoException`** to indicate enrollment failure, to request more captures or to veto enrollment (*page 73*).

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 SDK: Java Edition API functions used to perform the tasks in the workflow.

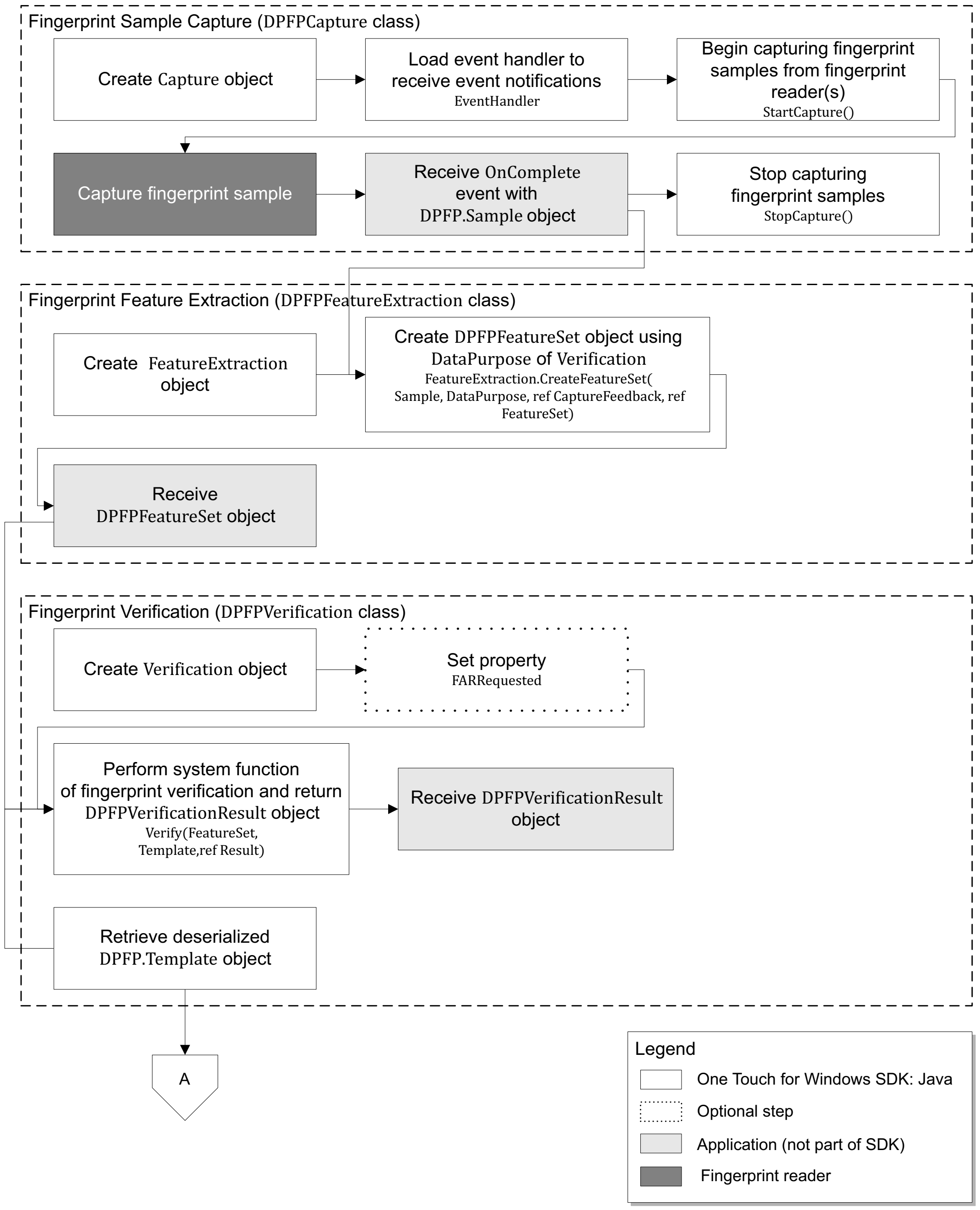

Fingerprint Sample Capture (DPFPCapture class)
Create Capture object
Load event handler to receive event notifications
EventHandler
Begin capturing fingerprint samples from fingerprint reader(s)
StartCapture()
Capture fingerprint sample
Receive OnComplete event with DPFP.Sample object
Stop capturing fingerprint samples
StopCapture()
Fingerprint Feature Extraction (DPFPFeatureExtraction class)
Create FeatureExtraction object
Create DPFPFeatureSet object using DataPurpose of Verification
FeatureExtraction.CreateFeatureSet(Sample, DataPurpose, ref CaptureFeedback, ref FeatureSet)
Receive DPFPFeatureSet object
Fingerprint Verification (DPFPVerification class)
Create Verification object
Set property
FARRequested
Perform system function of fingerprint verification and return DPFPVerificationResult object
Verify(FeatureSet, Template,ref Result)
Receive DPFPVerificationResult object
Retrieve deserialized DPFP.Template object
A
Legend
One Touch for Windows SDK: Java
Optional step
Application (not part of SDK)
Fingerprint reader

Figure 5. Typical fingerprint verification workflow

Fingerprint Sample Capture (DPFPCapture Class)

1. Create a new instance of the **`DPFPCapture`** class (*page 50*).

2. Load a fingerprint sample capture operation event handler for receiving event notifications by adding event listeners (*page 54*).

3. Begin capturing fingerprint samples from the fingerprint reader(s) connected to a system by calling the **`startCapture()`** method (*page 52*).

4. **Capture a fingerprint sample from a fingerprint reader. Note that you cannot change the priority or the reader(s) setting of a **`DPFPCapture`** object after it is started.

5. *Receive the **`dataAcquired()`** event from the fingerprint sample capture event handler along with a **`DPFPSample`** object when the fingerprint sample is successfully captured by the fingerprint reader (*page 54*).

6. *Pass the **`DPFPFeatureExtraction.createFeatureSet(DPFPSample sample, DPFPDataPurpose purpose);`** method. (See step 2 in the next section.)

7. Stop capturing fingerprint samples by calling the **`stopCapture`** method (*page 52*).

Fingerprint Feature Extraction (DPFPFeatureExtraction Class)

1. Create a new instance of the **`DPFPFeatureExtraction`** class (*page 65*).

2. Create **`DPFPFeatureSet`** objects by calling the **`createFeatureSet(DPFPSample sample, DPFPDataPurpose purpose)`** method using the value **`Verification`** for **`DataPurpose`** and passing the **`DPFPSample`** object from step 6 of the previous section (*page 65*).

3. *Pass the **`DPFPFeatureSet`** objects created in the previous step to the **`addFeatures`** method. (See step 2 in the next section.)

Fingerprint Verification (DPFPVerification Class)

1. Create a new instance of the **`DPFPVerification`** class (*page 81*).

   Example:

   ```
   DPFPVerification matcher = DPFPGlobal
     .getVerificationFactory()
     .createVerification();
   ```

2. Optionally, set the **`FARRequested`** property (*page 82*). You can use this property to set or to change the value of the FAR from the default or from a specified value.

   Example:

   ```
   matcher.setFARRequested(DPFPVerification.MEDIUM_SECURITY_FAR);
   ```

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

4. Create a **`DPFPTemplate`** object from the serialized data (see *Deserializing a Serialized Fingerprint Data Object* on *page 37*).

5. Perform the system function of fingerprint verification by calling the **`verify(featureSet, template)`** method and passing the **`DPFPTemplate`** object created in the previous step and the **`DPFPFeatureSet`** object from step 3 of the previous section (*page 82*).

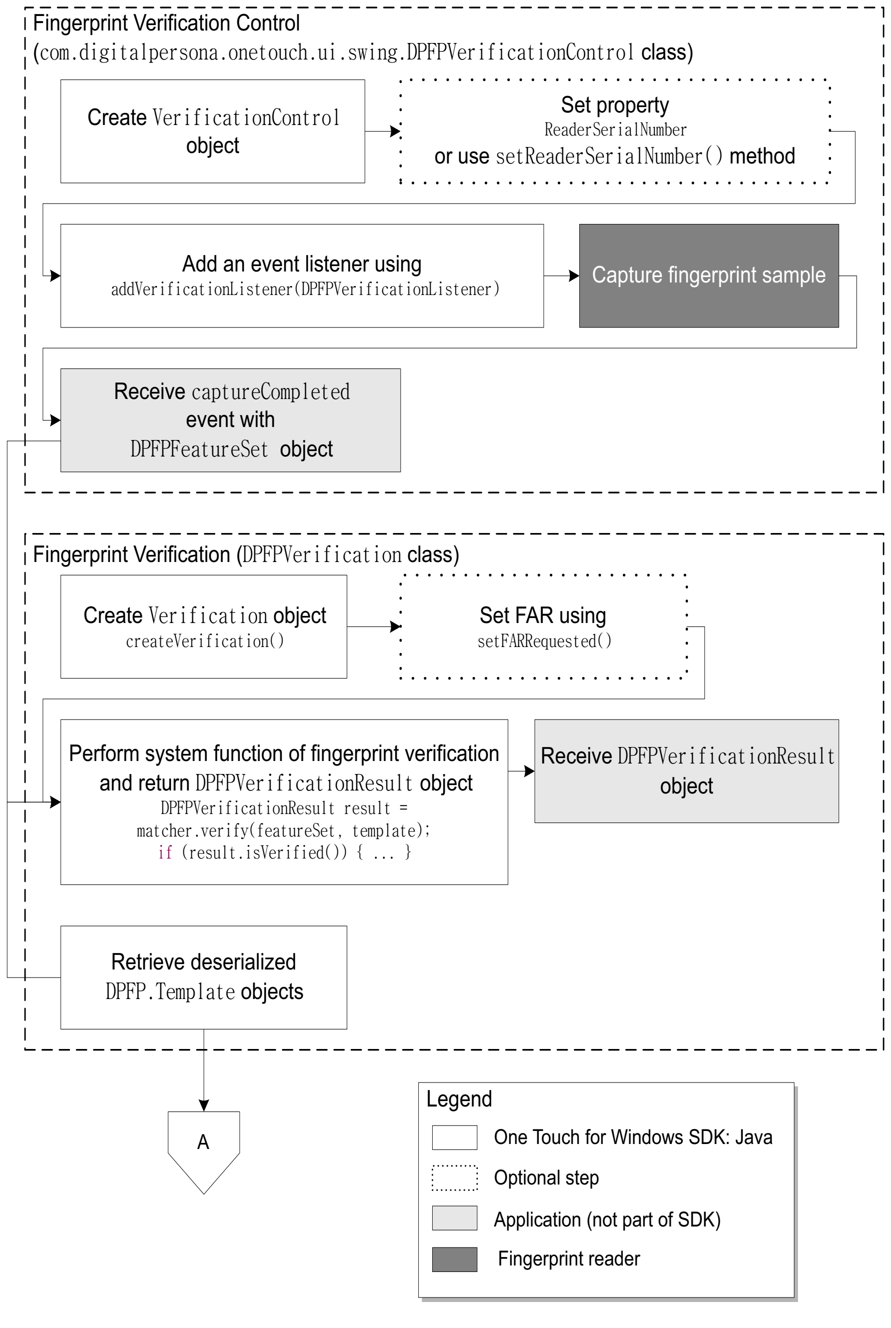
   Example:

   ```
   DPFPVerificationResult result = matcher.verify(featureSet, template);
   if (result.isVerified()) { ... }
   ```

6. *Receive the **`DPFPVerificationResult`** object, which provides the comparison decision of match or non-match (*page 83*).

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 SDK: Java Edition API functions used to perform the tasks in the workflow.



Fingerprint Verification Control
(com.digitalpersona.onetouch.ui.swing.DPFPVerificationControl class)
Create VerificationControl object
Set property ReaderSerialNumber or use setReaderSerialNumber() method
Add an event listener using addVerificationListener(DPFPVerificationListener)
Capture fingerprint sample
Receive captureCompleted event with DPFPFeatureSet object
Fingerprint Verification (DPFPVerification class)
Create Verification object createVerification()
Set FAR using setFARRequested()
Perform system function of fingerprint verification and return DPFPVerificationResult object
DPFPVerificationResult result = matcher.verify(featureSet, template); if (result.isVerified()) { ... }
Receive DPFPVerificationResult object
Retrieve deserialized DPFP.Template objects
A
Legend
One Touch for Windows SDK: Java
Optional step
Application (not part of SDK)
Fingerprint reader

Figure 6. Typical fingerprint verification with UI support workflow

Fingerprint Verification Control (DPFPVerificationControl Class)

1. Create a new instance of the **`DPFPVerificationControl`** class (*page 76*).
2. Optionally, set the **`ReaderSerialNumber`** property (*page 76*).
3. Add an event listener using **`addVerificationListener(DPFPVerificationListener)`** (*page 76*).
4. **Capture a fingerprint sample from a fingerprint reader.
5. Receive the **`captureCompleted`** event from the fingerprint verification control event handler along with the **`DPFPFeatureSet`** object (*page 80*).

Fingerprint Verification (DPFPVerification Class)

1. Create a new instance of the **`DPFPVerification`** class (*page 81*).

   Example:

   ```
   DPFPVerification matcher = DPFPGlobal
     .getVerificationFactory()
     .createVerification();
   ```
2. Optionally, set the **`FARRequested`** property (*page 82*). You can use this property to set or to change the value of the FAR from the default or from a specified value.

   Example:

   ```
   matcher.setFARRequested(DPFPVerification.MEDIUM_SECURITY_FAR);
   ```
3. *Retrieve serialized fingerprint template data from the fingerprint data storage subsystem.
4. Create a **`DPFPTemplate`** object from the serialized data (see *Deserializing a Serialized Fingerprint Data Object* on *page 37*).
5. Perform the system function of fingerprint verification by calling the **`verify(featureSet, template)`** method and passing the **`DPFPTemplate`** object created in the previous step and the **`DPFPFeatureSet`** object from step 3 of the previous section (*page 82*).

   Example:

   ```
   DPFPVerificationResult result = matcher.verify(featureSet, template);
   if (result.isVerified()) { ... }
   ```
6. *Receive the **`DPFPVerificationResult`** object, which provides the comparison decision of match or non-match (*page 83*).

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 SDK: Java Edition API functions used to perform the tasks in the workflows.

Serializing a Fingerprint Data Object

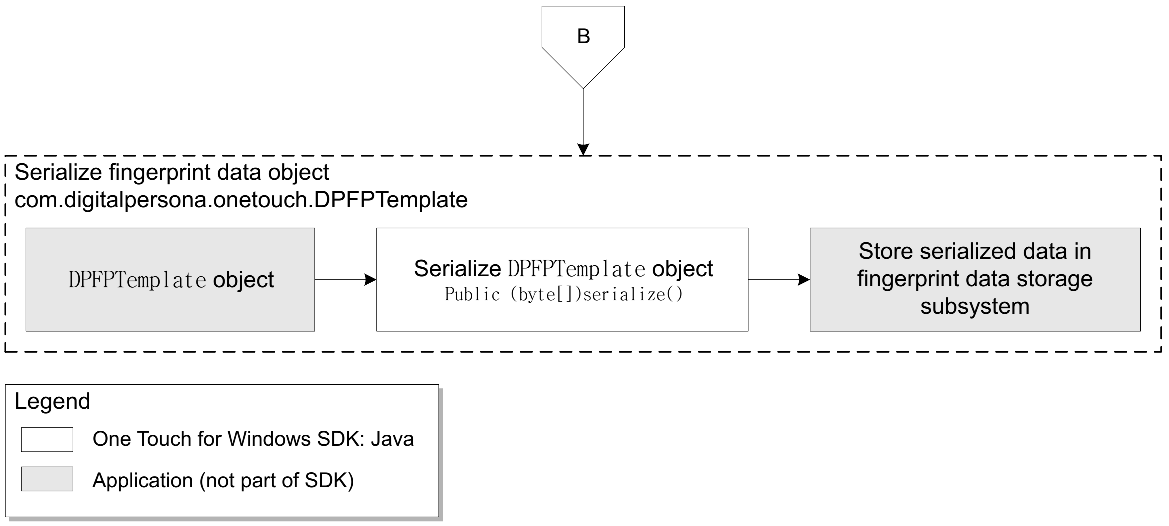

B
Serialize fingerprint data object
com.digitalpersona.onetouch.DPFPTemplate
DPFPTemplate object
Serialize DPFPTemplate object
Public (byte[])serialize()
Store serialized data in fingerprint data storage subsystem
Legend
One Touch for Windows SDK: Java
Application (not part of SDK)

Figure 7. Fingerprint data object serialization workflow: **`DPFPTemplate`** object

1. Begin with a **`DPFPTemplate`** object. (See *DPFPTemplate* on *page 47* for more information on how a **`DPFPTemplate`** object is constructed or supplied).
2. Serialize the **`DPFPTemplate`** object by calling the **`serialize`** method (*page 46*).
3. *Store the serialized fingerprint template data in a fingerprint data storage subsystem.

Deserializing a Serialized Fingerprint Data Object

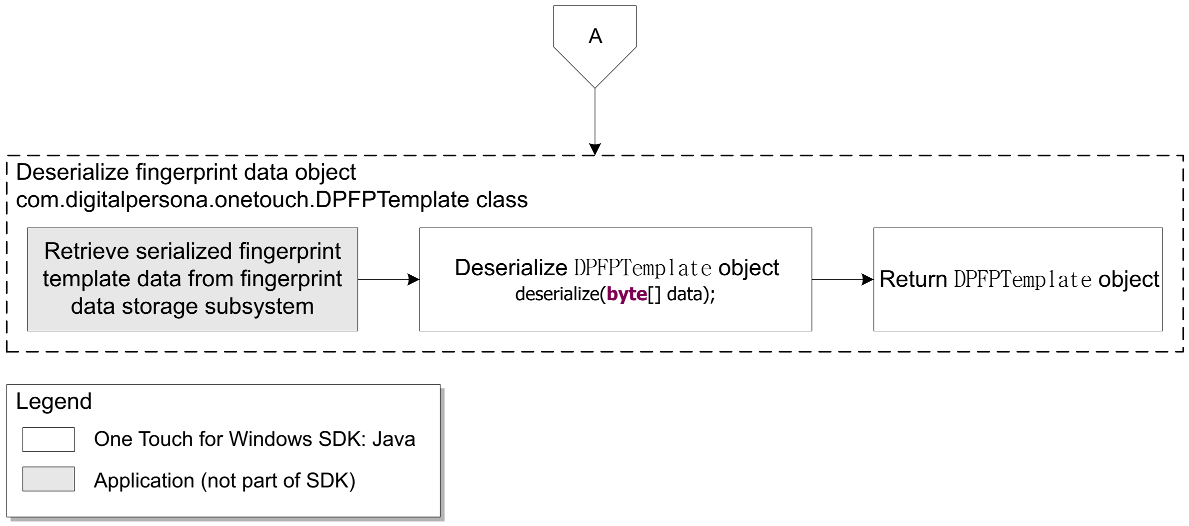

A
Deserialize fingerprint data object
com.digitalpersona.onetouch.DPFPTemplate class
Retrieve serialized fingerprint template data from fingerprint data storage subsystem
Deserialize DPFPTemplate object
deserialize(byte[] data);
Return DPFPTemplate object
Legend
One Touch for Windows SDK: Java
Application (not part of SDK)

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

1. *Retrieve serialized fingerprint template data from a fingerprint data storage subsystem.
2. Deserialize a **`DPFPTemplate`** object by calling the **`deserialize`** method (*page 46*).
3. Return a **`DPFPTemplate`** object.

API Reference 5

class onetouch

Type:	Package
Package:	digitalpersona

DPFPCaptureFeedback

Type:	Enumeration
Package:	onetouch
Description:	Feedback from capture operation.

Attributes

Attribute	Description
CAPTURE_FEEDBACK_GOOD Public «enum»	The sample is of good quality.
CAPTURE_FEEDBACK_NONE Public «enum»	There is no sample.
CAPTURE_FEEDBACK_TOO_LIGHT Public «enum»	The sample is too light.
CAPTURE_FEEDBACK_TOO_DARK Public «enum»	The sample is too dark.
CAPTURE_FEEDBACK_TOO_NOISY Public «enum»	The sample is too noisy.
CAPTURE_FEEDBACK_LOW_CONTRAST Public «enum»	The sample contrast is too low.
CAPTURE_FEEDBACK_NOT_ENOUGH_FEATURES Public «enum»	The sample does not contain enough information.

CAPTURE_FEEDBACK_NO_CENTRAL_REGION Public «enum»	The sample is not centered.
CAPTURE_FEEDBACK_NO_FINGER Public «enum»	The object scanned is not a finger.
CAPTURE_FEEDBACK_TOO_HIGH Public «enum»	The finger was too high on the swipe reader.
CAPTURE_FEEDBACK_TOO_LOW Public «enum»	The finger was too low on the swipe reader.
CAPTURE_FEEDBACK_TOO_LEFT Public «enum»	The finger was too close to the left border of swipe reader.
CAPTURE_FEEDBACK_TOO_RIGHT Public «enum»	The finger was too close to the right border of swipe reader.
CAPTURE_FEEDBACK_TOO_STRANGE Public «enum»	The scan looks strange.
CAPTURE_FEEDBACK_TOO_FAST Public «enum»	The finger was swiped too quickly.
CAPTURE_FEEDBACK_TOO_SKEWED Public «enum»	The image is too skewed.
CAPTURE_FEEDBACK_TOO_SHORT Public «enum»	The image is too short.
CAPTURE_FEEDBACK_TOO_SLOW Public «enum»	The finger was swiped too slowly.

DPFPDataPurpose

Type:	Enumeration
Package:	onetouch
Description:	Fingerprint data purpose.

Attributes

Attribute	Description
DATA_PURPOSE_UNKNOWN Public «enum»	The purpose of the data is not known.
DATA_PURPOSE_VERIFICATION Public «enum»	Fingerprint data will be used for the verification.
DATA_PURPOSE_ENROLLMENT Public «enum»	Fingerprint data will be used for the enrollment.

DPFPError

Type:	Class
Package:	onetouch
Description:	Describes the error structure.

Operations

Method	Description	Parameters
DPFPError() Protected	Constructs the error object.	**int** [in] errorCode The error code. **int** [in] extendedErrorCode The extended error code. **String** [in] errorText The error text. **Exception** [in] exception The exception caught.
DPFPError() Public	Constructs the error object.	**int** [in] errorCode The error code.

DPFPError() Public	Constructs the error object.	**int** [in] errorCode The error code. **String** [in] errorText The error text.
DPFPError() Public	Constructs the error object.	**int** [in] errorCode The error code. **int** [in] extendedErrorCode The extended error code.
DPFPError() Public	Constructs the error object.	**int** [in] errorCode The error code. **int** [in] extendedErrorCode The extended error code. **String** [in] errorText The error text.
DPFPError() Public	Constructs the error object.	**Exception** [in] exception The exception caught. **String** [in] errorText The error text.
DPFPError() Public	Constructs the error object.	**Exception** [in] exception The exception caught.
getErrorCode() int Public	Returns the error code.	
getErrorText() String Public	Returns the error text.	
getException() Exception Public	Returns the exception caught.	
getExtendedErrorCode() int Public	Returns the extended error code.	

DPFPFingerIndex

Type:	Enumeration
Package:	onetouch
Description:	Enumeration of fingers

Attributes

Attribute	Description
LEFT_PINKY Public «enum»	left little finger
LEFT_RING Public «enum»	left ring finger
LEFT_MIDDLE Public «enum»	left middle finger
LEFT_INDEX Public «enum»	left index finger
LEFT_THUMB Public «enum»	left thumb
RIGHT_THUMB Public «enum»	right thumb
RIGHT_INDEX Public «enum»	right index finger
RIGHT_MIDDLE Public «enum»	right middle finger
RIGHT_RING Public «enum»	right ring finger
RIGHT_PINKY Public «enum»	right little finger

Operations

Method	Description	Parameters
toBit() long Public	Returns the finger's bitmask. The left little finger corresponds to the least significant bit of the mask. The right little finger corresponds to the most significant bit of the mask.	
Static **fromMask()** EnumSet<DPFPFingerIndex> Public		**long** [in] mask

DPFPGlobal

Type:	Class
Package:	onetouch
Description:	This class represents a main access point to the OneTouch for Windows SDK: Java Edition library, and contains a set of static factory methods which allow the user to create various SDK classes.

Connections

Connector	Source	Target
Association Source -> Destination	Public DPFPGlobal	Private featureExtractionFactory DPFPFeatureExtractionFactory
Association Source -> Destination	Public DPFPGlobal	Private featureSetFactory DPFPFeatureSetFactory
Association Source -> Destination	Public DPFPGlobal	Private readersCollectionFactory DPFPReadersCollectionFactory
Association Source -> Destination	Public DPFPGlobal	Private sampleConversion DPFPSampleConversion
Association Source -> Destination	Public DPFPGlobal	Private templateFactory DPFPTemplateFactory
Association Source -> Destination	Public DPFPGlobal	Private captureFactory DPFPCaptureFactory
Association Source -> Destination	Public DPFPGlobal	Private enrollmentFactory DPFPEnrollmentFactory
Association Source -> Destination	Public DPFPGlobal	Private verificationFactory DPFPVerificationFactory
Association Source -> Destination	Public DPFPGlobal	Private sampleFactory DPFPSampleFactory

Operations

Method	Description	Parameters
Static **getCaptureFactory()** DPFPCaptureFactory Public	Returns the default factory for DPFPCapture objects.	
Static **getEnrollmentFactory()** DPFPEnrollmentFactory Public	Returns the default factory for DPFPEnrollment objects.	

Static **getFeatureExtractionFactory()** DPFPFeatureExtractionFactory Public	Returns the default factory for DPFPFeatureExtraction objects.
Static **getFeatureSetFactory()** DPFPFeatureSetFactory Public	Returns the default factory for DPFPFeatureSet objects.
Static **getReadersFactory()** DPFPReadersCollectionFactory Public	Returns the default factory for DPFPReadersCollection objects.
Static **getSampleConversionFactory()** DPFPSampleConversion Public	Returns the default factory for DPFPSampleConversion objects.
Static **getSampleFactory()** DPFPSampleFactory Public	Returns the default factory for DPFPSample objects.
Static **getTemplateFactory()** DPFPTemplateFactory Public	Returns the default factory for DPFPTemplate objects.
Static **getVerificationFactory()** DPFPVerificationFactory Public	Returns the default factory for DPFPVerification objects.

DPFPData

Type:	Interface
Package:	onetouch
Description:	Common structure of fingerprint data.

Operations

Method	Description	Parameters
deserialize() void Public	Imports the data from the binary representation.	**byte[]** [in] data The binary representation of a fingerprint object.
serialize() byte Public	Serializes opaque biometric data. Returns the binary representation of the fingerprint object.	

DPFPFeatureSet

Type:	Interface **DPFPData**
Package:	onetouch
Description:	The fingerprint feature set.

DPFPFeatureSetFactory

Type:	Interface
Package:	onetouch
Description:	DPFPFeatureSet factory interface.

Operations

Method	Description	Parameters
createFeatureSet() DPFPFeatureSet Public	Creates an empty DPFPFeatureSet object instance. Returns the object created.	
createFeatureSet() DPFPFeatureSet Public	Creates a DPFPFeatureSet object instance and fills it with data.	**byte[]** [in] data

DPFPSample

Type:	Interface **DPFPData**
Package:	onetouch
Description:	The fingerprint sample.

DPFPSampleFactory

Type:	Interface
Package:	onetouch
Description:	DPFPSample factory interface.

Operations

Method	Description	Parameters
createSample() DPFPSample Public	Creates an empty DPFPSample object instance. Returns the object created.	
createSample() DPFPSample Public	Creates an DPFPSample object instance and fills it with data. Returns the object created.	**byte[]** [in] data

DPFPTemplate

Type:	Interface DPFPData
Package:	onetouch
Description:	The fingerprint template.

DPFPTemplateFactory

Type:	Interface
Package:	onetouch
Description:	DPFPTemplate factory interface.

Operations

Method	Description	Parameters
createTemplate() DPFPTemplate Public	Creates an empty DPFPTemplate object instance. Returns the object created.	

createTemplate() DPFPTemplate Public	Creates a DPFPTemplate object instance and fills it with data. Returns the object created.	**byte[]** [in] data

capture package

Type:	**Package**
Package:	onetouch

DPFPCapturePriority

Type:	Enumeration
Package:	capture
Description:	Describes the priority of the fingerprint capture operation.

Attributes

Attribute	Description
CAPTURE_PRIORITY_LOW Public «enum»	Low priority. The subscriber uses this priority to acquire reader events only if there are no subscribers with high or normal priority. Only one subscriber with this priority is allowed.
CAPTURE_PRIORITY_NORMAL Public «enum»	Normal priority. The subscriber uses this priority to acquire device events only if the operation runs in a foreground process. Multiple subscribers with this priority are allowed.
CAPTURE_PRIORITY_HIGH Public «enum»	High priority. (RESERVED. For internal use only.) The subscriber uses this priority to acquire device events exclusively. Only one subscriber with this priority is allowed. The process subscribing with this priority must have administrative privileges or run under the Local SYSTEM account.

DPFPCapture

Type:	Interface
Package:	capture
Description:	This interface describes the operation of capturing fingerprint samples from a reader.

The capture operation subscribes to and monitors events on the selected fingerprint reader and notifies listeners about specific classes of events (image data and quality events, reader status events, reader events, errors).

One fingerprint reader may be monitored by several capture operations.

Each capture operation has a specific priority, which defines how reader events will be distributed among several concurrent operations.

Operations

Method	Description	Parameters
addDataListener() void Public	Adds the data event listener. See also DPFPDataEvent	**DPFPDataListener** [in] listener The listener to be added.
addErrorListener() void Public	Adds the error event listener. See also DPFPErrorEvent	**DPFPErrorListener** [in] listener The listener to be added.
addImageQualityListener() void Public	Adds the image quality event listener. See also DPFPImageQualityEvent	**DPFPImageQualityListener** [in] listener The listener to be added.
addReaderStatusListener() void Public	Adds the reader status event listener. See also DPFPReaderStatusEvent	**DPFPReaderStatusListener** [in] listener The listener to be added.
addSensorListener() void Public	Adds the sensor event listener. See also DPFPSensorEvent	**DPFPSensorListener** [in] listener The listener to be added.
getListeners() T Public	Enumerates all event listeners for the given class. Returns an array of event listeners.	**Class<T>** [in] t listener class.
getPriority() DPFPCapturePriority Public	Returns the current capture priority.	

getReaderSerialNumber() String Public	Returns the serial number of the fingerprint reader to be used for the capture.	
isStarted() boolean Public	Returns the status of the capture operation. Returns true if capture is started, false otherwise.	
removeDataListener() void Public	Removes the data event listener. See also DPFPDataEvent	**DPFPDataListener** [in] listener The listener to be removed.
removeErrorListener() void Public	Removes the error event listener. See also DPFPErrorEvent	**DPFPErrorListener** [in] listener The listener to be removed.
removeImageQualityListener() void Public	Removes the image quality event listener. See also DPFPImageQualityEvent	**DPFPImageQualityListener** [in] listener The listener to be removed.
removeReaderStatusListener() void Public	Removes the reader status event listener. See also DPFPReaderStatusEvent	**DPFPReaderStatusListener** [in] listener The listener to be removed.
removeSensorListener() void Public	Removes the sensor event listener. See also DPFPSensorEvent	**DPFPSensorListener** [in] listener The listener to be removed.
setPriority() void Public	Sets the capture priority. Modification of the priority is allowed only when the capture operation is not started, otherwise an IllegalStateException will be thrown.	**DPFPCapturePriority** [in] priority The capture priority.

setReaderSerialNumber() void Public	Sets the serial number of the fingerprint reader to be used for the capture. Modification of the serial number is allowed only when the capture operation is not started, otherwise an IllegalStateException will be thrown.	**String** [in] serialNumber The serial number of the fingerprint reader to be used for the capture.
startCapture() void Public	Starts the capture. The call is asynchronous and returns immediately. The events will be sent to the listeners until the stopCapture method is called.	
stopCapture() void Public	Stops the previously started capture operation.	

DPFPCaptureFactory

Type: Interface
Package: capture
Description: This interface describes a factory of DPFPCapture objects. Use one of the createCapture methods to construct a new DPFPCapture instance.

Operations

Method	Description	Parameters
createCapture() DPFPCapture Public	Creates a capture object. Returns the object created.	
createCapture() DPFPCapture Public	Creates a capture object on a specified reader. Returns the object created. See also DPFPReaderDescription and DPFPReadersCollection	**String** [in] readerSerialNumber A serial number of the specific fingerprint reader
createCapture() DPFPCapture Public	Creates a capture object with a specified priority. Returns the object created.	**DPFPCapturePriority** [in] priority
createCapture() DPFPCapture Public	Creates a capture object on a specified reader and with a specified priority. Returns the object created. See also DPFPReaderDescription and DPFPReadersCollection	**String** [in] readerSerialNumber **DPFPCapturePriority** [in] priority

event package

Type:	Package
Package:	capture

DPFPDataAdapter

Type:	Class
Package:	event
Description:	An abstract adapter class for receiving data events from the fingerprint capture device. The methods in this class are empty. This class exists as a convenience for creating listener objects.

Extend this class to create a DPFPDataEvent listener and override the methods for the events of interest. (If you implement the DPFPDataListener interface, you have to define all of the methods in it. This abstract class defines null methods for them all, so you only have to define methods for events you care about.)

Create a listener object using your class and then register it with a component using the component's addDataListener method.

Operations

Method	Description	Parameters
dataAcquired() void Public	Invoked when the fingerprint sample is acquired.	**DPFPDataEvent** [in] e The event occurred.

DPFPDataEvent

Type:	Class EventObject
Package:	event
Description:	An event indicating that a fingerprint has been acquired

Operations

Method	Description	Parameters
DPFPDataEvent() Public	Constructs an event.	**String** [in] readerSerialNumber The serial number of the reader on which the event initially occurred. **DPFPSample** [in] sample The fingerprint sample.
getSample() DPFPSample Public	Returns a fingerprint sample acquired.	

DPFPErrorAdapter

Type:	Class
Package:	event
Description:	An abstract adapter class for receiving error events from the fingerprint capture device. The methods in this class are empty. This class exists as a convenience for creating listener objects.

Extend this class to create a DPFPErrorEvent listener and override the methods for the events of interest. (If you implement the DPFPErrorListener interface, you have to define all of the methods in it. This abstract class defines null methods for them all, so you only have to define methods for events you care about.)

Create a listener object using your class and then register it with a component using the component's addErrorListener method.

Operations

Method	Description	Parameters
errorOccured() void Public	Fired when the error occurred.	**DPFPErrorEvent** [in] e The event occurred.
exceptionCaught() void Public	Fired when an exception caught.	**DPFPErrorEvent** [in] e The event occurred.

DPFPErrorEvent

Type:	Class EventObject
Package:	event
Description:	An error event occurred during a fingerprint capture.

Operations

Method	Description	Parameters
DPFPErrorEvent() Public	Constructs an event.	**String** [in] readerSerialNumber The serial number of the reader on which the event initially occurred. **DPFPError** [in] error The error occurred.
getError() DPFPError Public	Returns an error that occurred.	

DPFPImageQualityAdapter

Type:	Class
Package:	event
Description:	An abstract adapter class for receiving image quality feedback from the fingerprint capture device. The methods in this class are empty. This class exists as a convenience for creating listener objects.

Extend this class to create a DPFPImageQualityEvent listener and override the methods for the events of interest. (If you implement the DPFPImageQualityListener interface, you have to define all of the methods in it. This abstract class defines null methods for them all, so you only have to define methods for events you care about.)

Create a listener object using your class and then register it with a component using the component's addImageQualityListener method.

Operations

Method	Description	Parameters
onImageQuality() void Public	Invoked when the fingerprint sample is acquired with unsatisfactory quality.	**DPFPImageQualityEvent** [in] e The event occurred.

DPFPImageQualityEvent

Type:	Class EventObject
Package:	event
Description:	An event indicating that the quality of the fingerprint acquired is bad.

Operations

Method	Description	Parameters
DPFPImageQualityEvent() Public	Constructs an event.	**String** [in] readerSerialNumber The serial number of the reader on which the event initially occurred. **DPFPCaptureFeedback** [in] feedback The capture feedback.
getFeedback() DPFPCaptureFeedback Public	Returns feedback about the quality of the capture.	

DPFPReaderStatusAdapter

Type:	Class
Package:	event
Description:	An abstract adapter class for receiving reader status events from the fingerprint capture device.

The methods in this class are empty. This class exists as a convenience for creating listener objects.

Extend this class to create a DPFPReaderStatusEvent listener and override the methods for the events of interest. (If you implement the DPFPReaderStatusListener interface, you have to define all of the methods in it. This abstract class defines null methods for them all, so you only have to define methods for events you care about.)

Create a listener object using your class and then register it with a component using the component's addReaderStatusListener method.

Operations

Method	Description	Parameters
readerConnected() void Public	Invoked when the reader is connected.	**DPFPReaderStatusEvent** [in] e The event occurred.
readerDisconnected() void Public	Invoked when the reader is disconnected.	**DPFPReaderStatusEvent** [in] e The event occurred.

DPFPReaderStatusEvent

Type:	Class EventObject
Package:	event
Description:	An event indicating that a fingerprint reader has been connected or disconnected.

See also DPFPReaderStatusListener

Attributes

Attribute	Description	Constraints and tags
READER_CONNECTED int Public Static Const	Indicates that the reader has been connected. See also getReaderStatus()	Default*:* 3
READER_DISCONNECTED int Public Static Const	Indicates that the reader has been disconnected. See also getReaderStatus()	Default*:* 2

Operations

Method	Description	Parameters
DPFPReaderStatusEvent() Public	Constructs an event	**String** [in] readerSerialNumber The serial number of the reader on which the event initially occurred. **int** [in] readerStatus Either READER_CONNECTED or READER_DISCONNECTED.
getReaderStatus() int Public	Returns a reader status.	

DPFPSensorAdapter

Type:	Class
Package:	event
Description:	An abstract adapter class for receiving sensor events from the fingerprint capture device. The methods in this class are empty. This class exists as a convenience for creating listener objects.

Extend this class to create a DPFPSensorEvent listener and override the methods for the events of interest. (If you implement the DPFPSensorListener interface, you have to define all of the methods in it. This abstract class defines null methods for them all, so you only have to define methods for events you care about.)

Create a listener object using your class and then register it with a component using the component's addSensorListener method.

See also DPFPSensorEvent, DPFPSensorListener and DPFPCapture.

Operations

Method	Description	Parameters
fingerGone() void Public	Invoked when the finger removed from the reader.	**DPFPSensorEvent** [in] e The event occurred.
fingerTouched() void Public	Invoked when the finger put on the reader.	**DPFPSensorEvent** [in] e The event occurred.
imageAcquired() void Public	Invoked when the finger image is acquired and the finger can be removed from the reader.	**DPFPSensorEvent** [in] e The event occurred.

DPFPSensorEvent

Type:	Class EventObject
Package:	event
Description:	An event indicating an activity on a fingerprint reader. See also DPFPSensorListener.

Attributes

Attribute	Description	Constraints and tags
FINGER_GONE int Public Static Const	Indicates that the finger has been removed from the reader.	Default: 6
FINGER_TOUCH int Public Static Const	Indicates that the finger has been put on the reader. See also getSensorStatus().	Default: 5

IMAGE_READY int Public Static Const	Indicates that the image has been acquired and the finger can be removed from the reader.	Default: 7

Operations

Method	Description	Parameters
DPFPSensorEvent() Public	Constructs an event	**String** [in] readerSerialNumber A serial number of the reader on which the event initially occurred. **int** [in] sensorStatus Either FINGER_TOUCH, FINGER_GONE or IMAGE_READY.
getSensorStatus() int Public	Returns a sensor status.	

DPFPDataListener

Type:	Interface EventListener
Package:	event
Description:	The listener interface for receiving data events from a fingerprint reader. For the intermediate reader events (finger touch/gone) see DPFPSensorListener. See also DPFPDataEvent.

Operations

Method	Description	Parameters
dataAcquired() void Public	Invoked when a fingerprint sample is acquired.	**DPFPDataEvent** [in] e The event occurred.

DPFPErrorListener

Type:	Interface EventListener
Package:	event
Description:	The listener interface for receiving error events from a fingerprint reader. See also DPFPErrorEvent

Connections

Connector	Source	Target
Realisation Source -> Destination	Public DPFPErrorAdapter	Public DPFPErrorListener

Operations

Method	Description	Parameters
errorOccured() void Public	Fired when an error occurred.	**DPFPErrorEvent** [in] e The event that occurred.
exceptionCaught() void Public	Fired when an exception was caught.	**DPFPErrorEvent** [in] e The event that occurred.

DPFPImageQualityListener

Type:	Interface EventListener
Package:	event
Description:	The listener interface for receiving information about bad image quality. See also DPFPImageQualityEvent.

Operations

Method	Description	Parameters
onImageQuality() void Public	Invoked when the quality of the acquired fingerprint is unsatisfactory.	**DPFPImageQualityEvent** [in] e The event that occurred.

DPFPReaderStatusListener

Type:	Interface EventListener
Package:	event
Description:	The listener interface for receiving events when a fingerprint reader is connected or disconnected.

Operations

Method	Description	Parameters
readerConnected() void Public	Invoked when a reader is connected.	**DPFPReaderStatusEvent** [in] e The event occurred.
readerDisconnected() void Public	Invoked when a reader is disconnected.	**DPFPReaderStatusEvent** [in] e The event occurred.

DPFPSensorListener

Type:	Interface EventListener
Package:	event
Description:	The listener interface for receiving intermediate events from a fingerprint reader. For the reader data events see DPFPDataListener. See also DPFPSensorEvent and DPFPDataListener.

Operations

Method	Description	Parameters
fingerGone() void Public	Invoked when a finger is removed from a reader.	**DPFPSensorEvent** [in] e The event occurred.
fingerTouched() void Public	Invoked when a finger is put on a reader.	**DPFPSensorEvent** [in] e The event occurred.
imageAcquired() void Public	Invoked when a finger image is acquired and the finger can be removed from a reader.	**DPFPSensorEvent** [in] e The event occurred.

processing package

Type:	Package
Package:	onetouch

DPFPImageQualityException

Type:	Class Exception
Package:	processing
Description:	The exception will be thrown when the sample quality is not good enough for processing.

Operations

Method	Description	Parameters
DPFPImageQualityException() Public	Creates an exception with the given quality feedback.	**DPFPCaptureFeedback** [in] captureFeedback The capture quality feedback.
getCaptureFeedback() DPFPCaptureFeedback Public	Returns the capture feedback.	

DPFPTemplateStatus

Type:	Enumeration
Package:	processing
Description:	Status of the fingerprint template.

Attributes

Attribute	Description	Constraints and tags
TEMPLATE_STATUS_UNKNOWN Public «enum»	Status of the fingerprint template is unknown. Most probably the fingerprint template does not exist yet.	Default:
TEMPLATE_STATUS_INSUFFICIENT Public «enum»	The fingerprint template exists, but more fingerprint samples are required to finalize it.	Default:
TEMPLATE_STATUS_FAILED Public «enum»	The creation of the fingerprint template failed.	Default:
TEMPLATE_STATUS_READY Public «enum»	Fingerprint template was successfully created.	Default:

DPFPEnrollment

Type:	Interface
Package:	processing
Description:	Creates a fingerprint template from a number of fingerprint feature sets.

Operations

Method	Description	Parameters
addFeatures() void Public	Adds a new fingerprint feature set to the source data collection.	**DPFPFeatureSet** [in] featureSet A fingerprint feature set to be added.
clear() void Public	Clears the source data collection and resets the result fingerprint template status to DPFPTemplateStatus#TEMPLATE_STATUS_UNKNOWN. The caller can start template creation over again.	
getFeaturesNeeded() int Public	Returns the number of fingerprint feature sets needed to create the fingerprint template. This value decreases as each feature set is added, showing the actual number of feature sets still needed in order to complete creation of the fingerprint template.	
getTemplate() DPFPTemplate Public	Returns the fingerprint template created.	
getTemplateStatus() DPFPTemplateStatus Public	Returns the status of the fingerprint template.	

DPFPEnrollmentFactory

Type:	Interface
Package:	processing
Description:	DPFPEnrollment factory interface.

Operations

Method	Description	Parameters
createEnrollment() DPFPEnrollment Public	Creates an object implementing DPFPEnrollment interface.	

DPFPFeatureExtraction

Type:	Interface
Package:	processing
Description:	This interface represents a fingerprint feature extractor.

The fingerprint feature extractor converts a sample captured from a fingerprint reader into a set of fingerprint features, unique for the fingerprint and specific for any of following usages (purposes): enrollment or verification. See also DPFPSample and DPFPDataPurpose.

Operations

Method	Description	Parameters
createFeatureSet() DPFPFeatureSet Public	Extracts a purpose-specific fingerprint feature set from the fingerprint sample. Returns the fingerprint feature set created.	**DPFPSample** [in] sample The source fingerprint sample. **DPFPDataPurpose** [in] purpose The purpose of the feature set.

DPFPFeatureExtractionFactory

Type:	Interface
Package:	processing
Description:	DPFPFeatureExtraction factory interface.

Operations

Method	Description
createFeatureExtraction() DPFPFeatureExtraction Public	Creates an object implementing DPFPFeatureExtraction interface. Returns the object created.

DPFPSampleConversion

Type:	Interface
Package:	processing
Description:	This interface provides converters for fingerprint sample data.

Once a fingerprint sample has been captured from a reader, it is possible to convert it to other formats: an image or an ANSI 381compliant binary representation.

Operations

Method	Description	Parameters
convertToAnsi381() byte Public	Converts the sample to an ANSI 381 compliant format.	**DPFPSample** [in] sample The fingerprint sample.
createImage() Image Public	Converts the sample to a standard Java image.	**DPFPSample** [in] sample The fingerprint sample.

readers package

Type: Package
Package: onetouch

DPFPReaderImpressionType

Type: Enumeration
Package: readers
Description: Describes the fingerprint reader impression type.

Attributes

Attribute	Description	Constraints and tags
READER_IMPRESSION_TYPE_UNKNOWN Public «enum»	The fingerprint reader impression type is unknown.	Default:
READER_IMPRESSION_TYPE_SWIPE Public «enum»	The reader scans fingerprints with a swiping method.	Default:
READER_IMPRESSION_TYPE_AREA Public «enum»	The reader scans fingerprints with a touching method.	Default:

DPFPReaderSerialNumberType

Type: Enumeration
Package: readers
Description: Describes whether the serial number of the fingerprint reader is provided by hardware or software.

Attributes

Attribute	Description	Constraints and tags
SERIAL_NUMBER_TYPE_PERSISTENT Public «enum»	The persistent serial number of the fingerprint reader is provided by hardware.	Default:
SERIAL_NUMBER_TYPE_VOLATILE Public «enum»	The volatile serial number of the fingerprint reader is provided by software.	Default:

DPFPReaderTechnology

Type: Enumeration
Package: readers
Description: Describes the fingerprint reader technology.

Attributes

Attribute	Description	Constraints and tags
READER_TECHNOLOGY_UNKNOWN Public «enum»	The fingerprint reader technology is unknown.	Default:
READER_TECHNOLOGY_OPTICAL Public «enum»	Optical fingerprint reader.	Default:
READER_TECHNOLOGY_CAPACITIVE Public «enum»	Capacitive fingerprint reader.	Default:
READER_TECHNOLOGY_THERMAL Public «enum»	Thermal fingerprint reader.	Default:
READER_TECHNOLOGY_PRESSURE Public «enum»	Pressure fingerprint reader.	Default:

DPFPReaderDescription

Type:	Interface
Package:	readers
Description:	Provides information about a particular physical fingerprint reader attached to the system.

Operations

Method	Description	Parameters
getFirmwareRevision() DPFPReaderVersion Public	Returns the fingerprint reader firmware revision.	
getHardwareRevision() DPFPReaderVersion Public	Returns the fingerprint reader hardware revision.	
getImpressionType() DPFPReaderImpressionType Public	Returns the fingerprint reader usage modality.	
getLanguage() int Public	Returns the fingerprint reader language.	
getProductName() String Public	Returns the fingerprint reader product class name.	
getSerialNumber() String Public	Returns the fingerprint reader serial number.	
getSerialNumberType() DPFPReaderSerialNumberType Public	Describes if the fingerprint reader serial number is provided by hardware or software.	
getTechnology() DPFPReaderTechnology Public	Returns the fingerprint reader technology.	
getVendor() String Public	Returns the fingerprint reader manufacturer name.	

DPFPReaderVersion

Type:	Interface
Package:	readers
Description:	The fingerprint reader version information.

Operations

Method	Description
getBuild() int Public	Returns the build number of the reader.
getMajor() int Public	Returns the major version of the reader.
getMinor() int Public	Returns the minor version of the reader.

DPFPReadersCollection

Type:	Interface List
Package:	readers
Description:	Collection of the descriptions of all fingerprint readers attached to the system.

Operations

Method	Description	Parameters
get() DPFPReaderDescription Public	Finds a description of the fingerprint reader by its serial number. Returns the description found or null if nothing was found.	**String** [in] serialNumber The serial number.

DPFPReadersCollectionFactory

Type:	Interface
Package:	readers
Description:	DPFPReadersCollection factory interface.

Operations

Method	Description
getReaders() DPFPReadersCollection Public	Creates a new list of available reader descriptions. Returns the object created.

swing package

Type: Package
Package: ui

DPFPEnrollmentControl

Type: Class JPanel, Serializable
Package: swing
Description Swing UI for Enrollment

Attributes

Attribute	Description	Constraints and tags
DPFPUI_PROPERTIES String Package Static Const		Default: "dpfpui"
ENROLLED_FINGERS_PROPERTY String Package Static Const		Default: "enrolledFingers"
MAX_ENROLLED_FINGER_COUNT String Package Static Const		Default: "maxEnrollFingerCount"
PREFFERED_HEIGHT int Package Static Const		Default: 320
PREFFERED_WIDTH int Package Static Const		Default: 480
READER_SERIAL_NUMBER_PROPERTY String Package Static Const		Default: "readerSerialNumber"

Operations

Method	Description	Parameters
addEnrollmentListener() void Public	Adds enrollment event listener	**DPFPEnrollmentListener** [in] listener listener to add
DPFPEnrollmentControl() Public	Creates enrollment control.	

getEnrolledFingers() EnumSet<DPFPFingerIndex> Public	Returns the enrolled finger indices	
getMaxEnrollFingerCount() int Public	Returns the maximum number of fingers allowed for the user	
getReaderSerialNumber() String Public	Returns the serial number of fingerprint reader from which data will be captured.	
removeEnrollmentListener() void Public	Removes enrollment event listener	**DPFPEnrollmentListener** [in] listener listener to remove
setEnrolledFingers() void Public	Sets the enrolled finger indices	**EnumSet<DPFPFingerIndex>** [in] fingers fingers enrolled
setMaxEnrollFingerCount() void Public	Sets the maximum number of fingers allowed for the user	**int** [in] maxCount maximum number to set
setReaderSerialNumber() void Public	Sets the serial number of fingerprint reader from which data will be captured. Setting the serial number will stop capturing.	**String** [in] serialNumber serial number to be set

DPFPEnrollmentEvent

Type:	Class EventObject
Package:	swing
Description:	Event occurred as a result of user input in the enrollment control.

Attributes

Attribute	Description	Constraints and tags
FINGER_DELETED int Public Static Const	Indicates that finger enrollment is deleted and template should be removed	*Default:* 1
FINGER_ENROLLED int Public Static Const	Indicates that finger is enrolled and needs to be saved See also getID()	*Default:* 0

Operations

Method	Description	Parameters
DPFPEnrollmentEvent() Public	Constructs an event	**Object** [in] source event source **int** [in] id event id **DPFPFingerIndex** [in] finger finger index **DPFPTemplate** [in] template enrolled template
DPFPEnrollmentEvent() Public	Constructs an event with null template	**Object** [in] source event source **int** [in] id event id **DPFPFingerIndex** [in] finger finger index
getFingerIndex() DPFPFingerIndex Public	Returns the finger index associated with the event.	
getID() int Public	Returns the event ID.	

getPerformed() boolean Public		
getStopCapture() boolean Public		
getTemplate() DPFPTemplate Public	Returns the enrolled fingerprint template.	
setPerformed() void Public	Sets an enrollment or unenrollment result to report in the enrollment UI control. Event handler may signal the success of either one by setting the performed property to true. Setting the property to true or false doesn't prevent other listeners from receiving the event - all listeners will be notified and may modify the property again. So, if there are several listeners subscribed to the event, they can combine their results using either AND ("any failure means unsuccessful [un]enrollment") or OR ("any success means successful [un]enrollment"). After notifying all listeners, the control will show the resulting [un]enrollment status. If some listener wants to signal [un]enrollment results immediately, it can set the event's properties and throw a DPFPEnrollmentVetoException. In this case all remaining listeners will be skipped and [un]enrollment status will be reflected immediately.	**boolean** [in] performed True signals that enrollment or unenrollment was successfully performed (default), False signals failure to perform the [un]enrollment. **boolean** [in] stop True signals that there is no need for new captures and the capture operation may be stopped (default). False signals that the enrollment needs more fingerprint captures.

setStopCapture() void Public	Sets a request for a capture cancellation. By default, when a fingerprint is captured and all listeners are notified, the capture operation stops. If some event listener wants to repeat capture, it may set the stopCapture property to false, then after notifying all listeners the capture will continue. If some listener wants to signal enrollment result and stop capture immediately, it can set the event's properties and throw a DPFPEnrollmentVetoException. In this case all remaining listeners will be skipped, capture will stop and enrollment status will be reflected immediately.

DPFPEnrollmentVetoException

Type:	Class Exception
Package:	swing
Description:	Can be thrown by DPFPEnrollment listeners in order to signal that results of [un]enrollment should be applied immediately, ignoring other listeners.

See also DPFPEnrollmentEvent and DPFPEnrollmentListener.

Operations

Method	Description	Parameters
DPFPEnrollmentVetoException() Public	Constructs a new exception.	
DPFPEnrollmentVetoException() Public		**String** [in] reason

DPFPVerificationControl

Type:	Class JPanel
Package:	swing
Description:	This class implements a Java Swing UI control for verification.

Attributes

Attribute	Description	Constraints and tags
READER_SERIAL_NUMBER_PROPERTY String Public Static Const		Default: "readerSerialNumber"

Operations

Method	Description	Parameters
addVerificationListener() void Public	Adds a verification event listener. See also DPFPVerificationEvent and DPFPVerificationListener.	**DPFPVerificationListener** [in] listener the listener to add
DPFPVerificationControl() Public	Constructs a new verification UI control.	
getReaderSerialNumber() String Public	Returns the serial number of the fingerprint reader from which data will be captured.	
isStopping() boolean Public	Indicates that the control has a pending request to stop. Returns true when the control has a pending request to stop, false otherwise.	
removeVerificationListener() void Public	Removes a verification event listener. See also DPFPVerificationEvent and DPFPVerificationListener.	**DPFPVerificationListener** [in] listener the listener to remove
setReaderSerialNumber() void Public	Sets the serial number of the fingerprint reader from which data will be captured. Note that setting the serial number will stop capturing.	**String** [in] serialNumber the serial number to be set

start() void Public	Starts a fingerprint capture. This method makes the control begin waiting for a fingerprint capture from the reader. When the fingerprint is obtained, the control creates a feature set and generates a DPFPVerificationEvent. Event listeners may compare this feature set with their fingerprint templates and report success/failure to the control using the PFPVerificationEvent#setMatched property. When all listeners are notified, the control will show a matching status and stop capturing. If some listener wants more captures, it can set DPFPVerificationEvent#getStopCapture() to false (by default it is set to true). If the DPFPVerificationVetoException has been thrown by any listener, then remaining listeners will not receive the captureCompleted event, and the control immediately shows matching status and stops capture (unless the DPFPVerificationEvent#getStopCapture() property was reset to false).	
stop() void Public	Places a request to stop capture. The stopping is asynchronous due to JNI and Swing thread interlocking. After the request is placed, all events from the capture must be ignored - check the iStopping() flag.	

DPFPVerificationEvent

Type:	Class EventObject
Package:	swing
Description:	This class represents an event which occurs in a verification UI control as a result of user action.

Event listeners can modify "status" properties of the event setMatched, setStopCapture thus reporting about verification results or requesting more fingerprint captures.

Operations

Method	Description	Parameters
DPFPVerificationEvent() Public	Constructs an event	**Object** [in] source event source **DPFPFeatureSet** [in] featureSet acquired fingerprint feature set
getFeatureSet() DPFPFeatureSet Public	Returns acquired fingerprint feature set	
getMatched() boolean Public		
getStopCapture() boolean Public		

setMatched() void Public	Sets a verification result which is reported in the verification control. Event handlers may signal a verification failure setting the matched property into true. Setting the property to true or false doesn't prevent other listeners from receiving the event, all listeners will be notified and may modify the property again. So, if there are several listeners subscribed to the event, they can combine their results using either AND ("any failure means unsuccessful match") or OR ("any success means successful match"). After notifying all listeners, the control will show the resulting verification status. If some listener wants to signal verification result immediately, it can set the event's properties and throw a DPFPVerificationVetoException. In this case all remaining listeners will be skipped and verification status will be reflected immediately. Default value is false.	**boolean** [in] matched True signals that verification was successfully matched . False signals failure to find any matching template. (default)
setStopCapture() void Public	Sets a request for a capture cancellation. By default, when a fingerprint is captured and all listeners are notified, the capture operation stops. If some event listener wants to repeat capture, it may set the stopCapture property to false, then after notifying all listeners the capture will continue. If some listener wants to signal verification result and stop capture immediately, it can set the event's properties and throw a DPFPVerificationVetoException. In this case all remaining listeners will be skipped, capture will stop and verification status will be reflected immediately.	**boolean** [in] stop True signals that there is no need for new captures and that the capture operation may be stopped (default),. False signals that the verification needs more fingerprint captures.

DPFPVerificationVetoException

Type: Class Exception

Package:	swing
Description:	Can be thrown by DPFPVerification listeners in order to signal that the capturing should be continued.

Operations

Method	Description	Parameters
DPFPVerificationVetoException() Public	Constructs a new exception.	

DPFPEnrollmentListener

Type:	Interface EventListener
Package:	swing
Description:	Listener interface for enrollment control events.

Operations

Method	Description	Parameters
fingerDeleted() void Public	Fired when old fingerprint needs to be deleted.	**DPFPEnrollmentEvent** [in] e event occurred
fingerEnrolled() void Public	Fired when new fingerprint was enrolled and needs to be saved.	**DPFPEnrollmentEvent** [in] e event occurred

DPFPVerificationListener

Type:	Interface EventListener
Package:	swing
Description:	Listener interface for the verification UI control events.

Operations

Method	Description	Parameters
captureCompleted() void Public	Fired when fingerprint has been acquired and verification feature set is extracted.	**DPFPVerificationEvent** [in] e The verification event that occurred. Listeners should modify the event to signal a verification result.

verification package

Type:	Package
Package:	onetouch

DPFPVerification

Type:	Interface
Package:	verification
Description:	This interface describes a fingerprint verification operation.

The fingerprint verification interface allows comparison of a fingerprint feature set (extracted from a fingerprint reader capture) with a fingerprint template (created during fingerprint enrollment). It uses a false accept rate (FAR) as a threshold to decide whether or not the feature set and template match each other close enough to accept the feature set. Lower FAR means lower probability of falsely accepted fingerprint but higher rate of false rejects (FRR); and vice versa.
See also DPFPCapture, DPFPFeatureExtraction and DPFPEnrollment.

Attributes

Attribute	Description	Constraints and tags
HIGH_SECURITY_FAR int Public Static Const	False accept rate (FAR) factor corresponding to false accept probability of 0.000001 (1e-6) (most strict verification)	Default: PROBABILITY_ONE / 1000000
LOW_SECURITY_FAR int Public Static Const	False accept rate (FAR) factor corresponding to false accept probability of 0.0001 (1e-4) (not very strict verification)	Default: PROBABILITY_ONE / 10000
MEDIUM_SECURITY_FAR int Public Static Const	False accept rate (FAR) factor corresponding to false accept probability of 0.00001 (1e-5) (moderately strict verification)	Default: PROBABILITY_ONE / 100000
PROBABILITY_ONE int Public Static Const	False accept rate (FAR) corresponding to false accept probability of 1.0 (every fingerprint template will successfully match).	*Default:* 0x7FFFFFFF

Operations

Method	Description	Parameters

getFARRequested() int Public	Returns the false accept rate (FAR) factor set for the operation. Corresponding probability (normalized to [0..1]) of false accept may be estimated as: double p = ((double)getFARRequested()) / PROBABILITY_ONE;. Lower FAR means lower probability of falsely accepted fingerprint but higher rate of false rejects (FRR); and vice versa.	
setFARRequested() void Public	Sets the false accept rate (FAR) factor. Corresponding probability (normalized to [0..1]) of false accept may be estimated as: double p = ((double)farRequested()) / PROBABILITY_ONE. So, having the desired normalized probability p, you can calculate the farRequested value as: int farRequested = p * PROBABILITY_ONE. Lower FAR means lower probability of falsely accepted fingerprint but higher rate of false rejects (FRR); and vice versa.	**int** [in] farRequested The false accept rate (FAR) factor requested.
verify() DPFPVerificationResult Public	Compares the fingerprint feature set against the fingerprint template and returns the result of comparison. The fingerprint verification uses a false accept rate (FAR) as a threshold to decide are the feature set and template match each other close enough to accept the feature set. See also getFARRequested()	**DPFPFeatureSet** [in] featureSet Fingerprint feature set to verify. **DPFPTemplate** [in] enrollmentTemplate Fingerprint template to verify against.

DPFPVerificationFactory

Type:	Interface
Package:	verification
Description:	DPFPVerification factory interface.

Operations

Method	Description	Parameters
createVerification() DPFPVerification Public	Creates an instance of DPFPVerification object.	
createVerification() DPFPVerification Public	Creates an instance of DPFPVerification object with a specified "false accept ratio" value.	**int** [in] FARRequested

DPFPVerificationResult

Type:	Interface
Package:	verification
Description:	This interface represents a result of verification operation.

Operations

Method	Description
getFalseAcceptRate() int Public	Returns a value indicating the verification score; a number signifying how closely the fingerprint and template match each other). See also getFARRequested
isVerified() boolean Public	Returns the decision: whether or not a fingerprint feature set matches the fingerprint template closely enough. Returns true if fingerprint feature set matches the fingerprint template, false otherwise.

Graphical User Interfaces 6

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

- **`com.digitalpersona.onetouch.ui.swing.DPFPEnrollmentControl`**

  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 71*.

- **`com.digitalpersona.onetouch.ui.swing.DPFPVerificationControl`**

  This object includes the graphical user interface described on *page 93*. The constructor, properties, and event handler contained within this namespace are described on *page 76*.

DPFPEnrollmentControl User Interface

The graphical user interface included with the **`DPFPEnrollmentControl`** 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 1 illustrates the fingerprint enrollment process using the **`DPFPEnrollmentControl`** object graphical user interface. Picture numbers in the figure correspond to the pictures in Table 1 on *page 86*. *Table 1* illustrates and describes the interaction between the user and the graphical user interface during fingerprint enrollment.

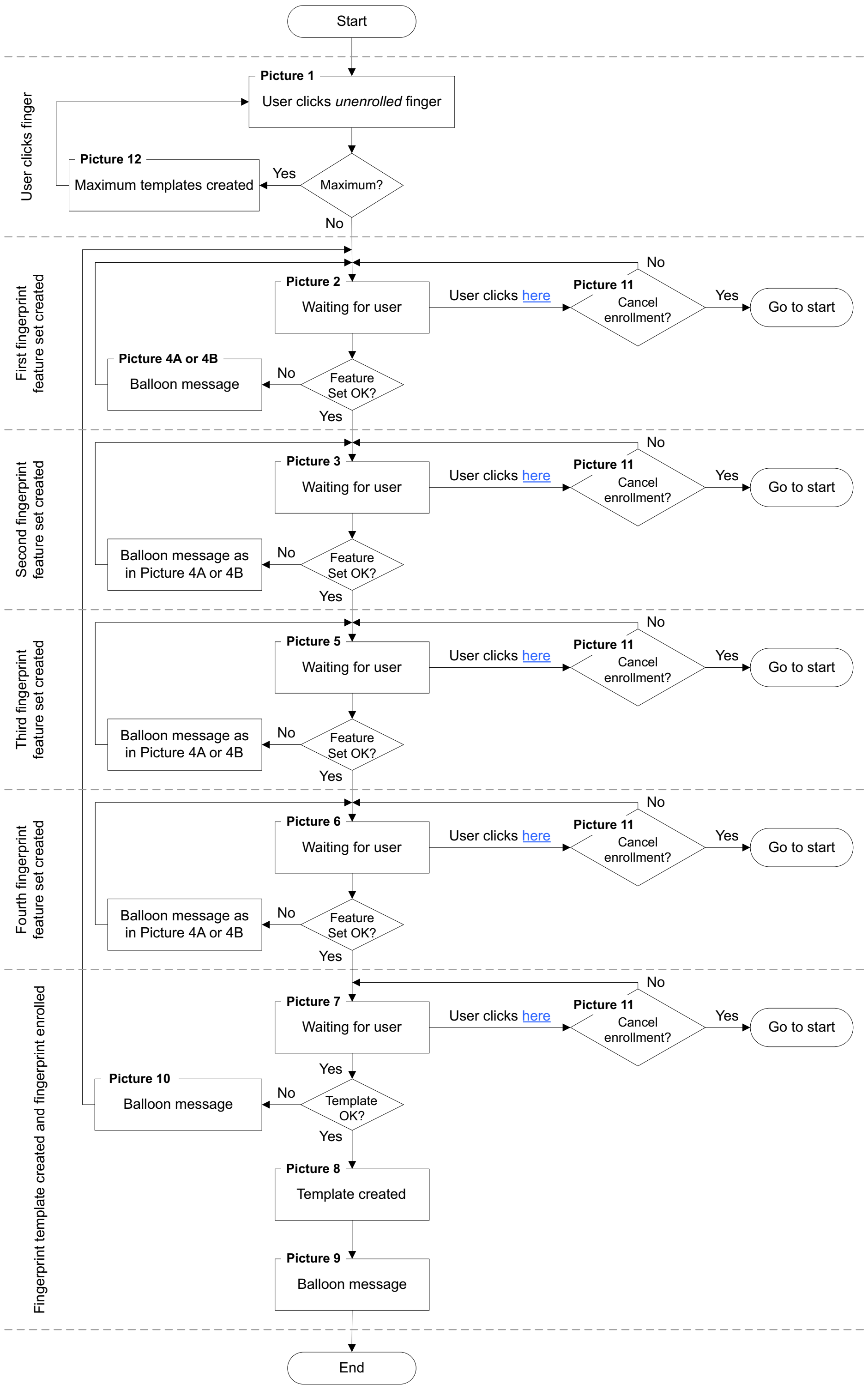


Figure 1. Enrolling a fingerprint using the `DPFPEnrollmentControl` object graphical user interface

Table 1. `DPFPEnrollmentControl` object graphical user interface: Enrolling a fingerprint

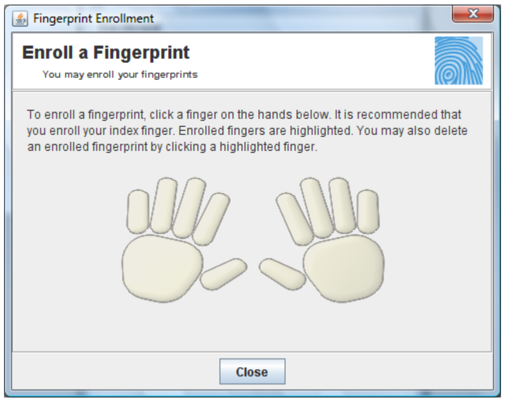
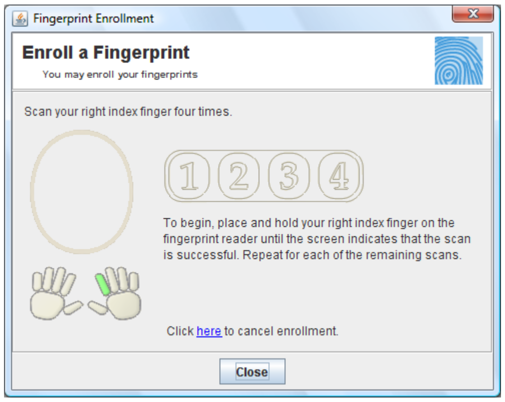
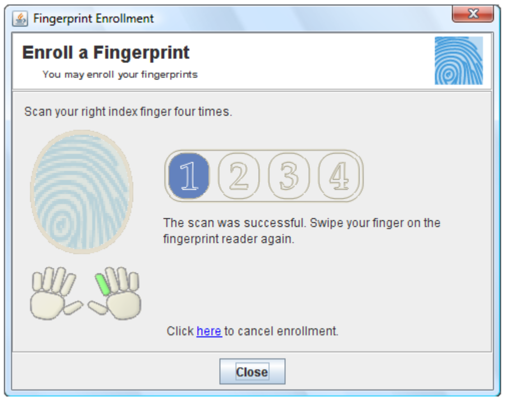
Graphical user interface	**User actions and user interface feedback**
Picture 1  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints To enroll a fingerprint, click a finger on the hands below. It is recommended that you enroll your index finger. Enrolled fingers are highlighted. You may also delete an enrolled fingerprint by clicking a highlighted finger.	This image indicates that no fingerprints have been enrolled, because the fingers associated with any enrolled fingerprints are green.
Picture 2  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints Scan your right index finger four times. To begin, place and hold your right index finger on the fingerprint reader until the screen indicates that the scan is successful. Repeat for each of the remaining scans. Click here to cancel enrollment.	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  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints Scan your right index finger four times. The scan was successful. Swipe your finger on the fingerprint reader again. Click here to cancel enrollment.	The user touched the fingerprint reader, and a fingerprint feature set was created.

Table 1. `DPFPEnrollmentControl` object graphical user interface: Enrolling a fingerprint *(continued)*

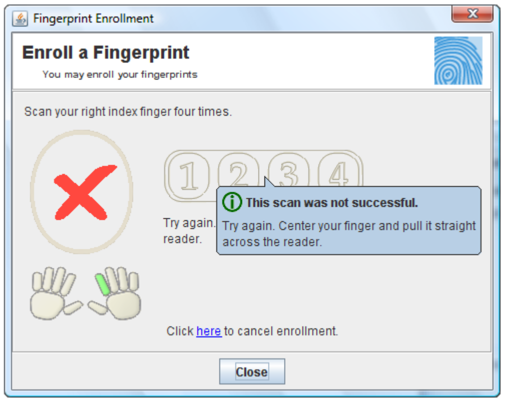
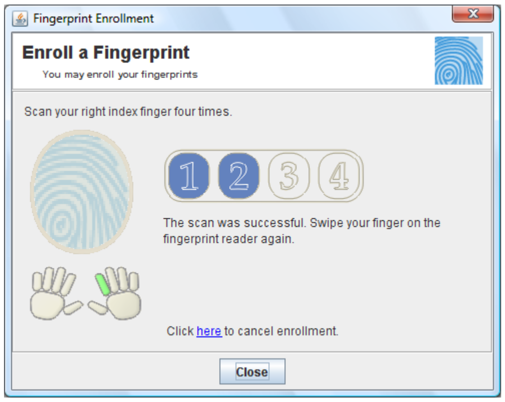
Graphical user interface	**User actions and user interface feedback**
Picture 4  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints Scan your right index finger four times. **This scan was not successful.** Try again. Center your finger and pull it straight across the reader. Try again. reader. Click here to cancel enrollment. 	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.
Picture 5  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints Scan your right index finger four times. The scan was successful. Swipe your finger on the fingerprint reader again. Click here to cancel enrollment. 	The user touched the fingerprint reader, and a second fingerprint feature set was created.

Table 1. `DPFPEnrollmentControl` object graphical user interface: Enrolling a fingerprint *(continued)*

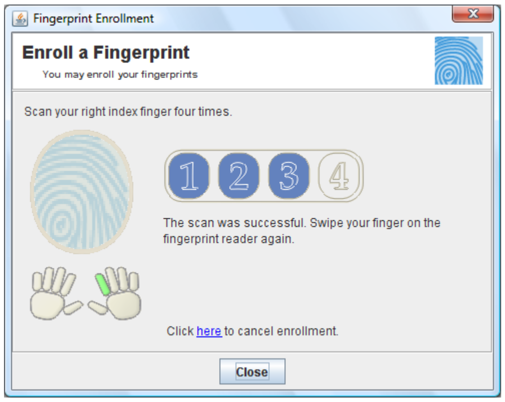
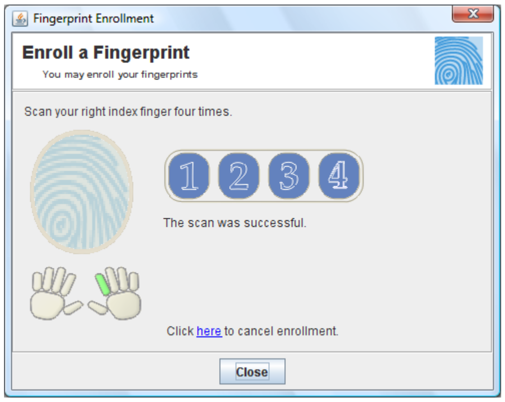
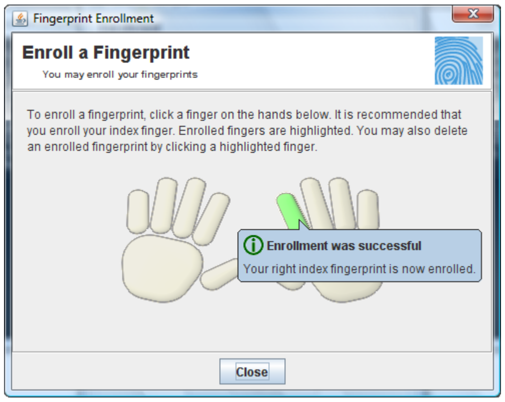
Graphical user interface	User actions and user interface feedback
Picture 6  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints Scan your right index finger four times. The scan was successful. Swipe your finger on the fingerprint reader again. Click here to cancel enrollment.	The user touched the fingerprint reader, and a third fingerprint feature set was created.
Picture 7  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints Scan your right index finger four times. The scan was successful. Click here to cancel enrollment.	The user touched the fingerprint reader, and a fourth fingerprint feature set was created.
Picture 8  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints To enroll a fingerprint, click a finger on the hands below. It is recommended that you enroll your index finger. Enrolled fingers are highlighted. You may also delete an enrolled fingerprint by clicking a highlighted finger. Enrollment was successful Your right index fingerprint is now enrolled.	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 1. `DPFPEnrollmentControl` object graphical user interface: Enrolling a fingerprint *(continued)*

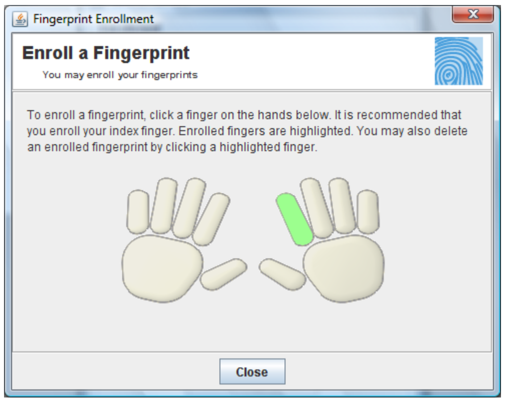
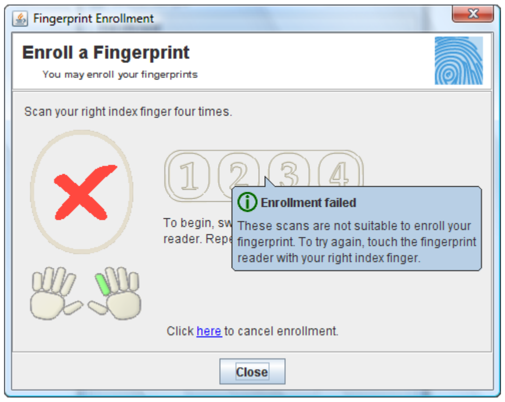
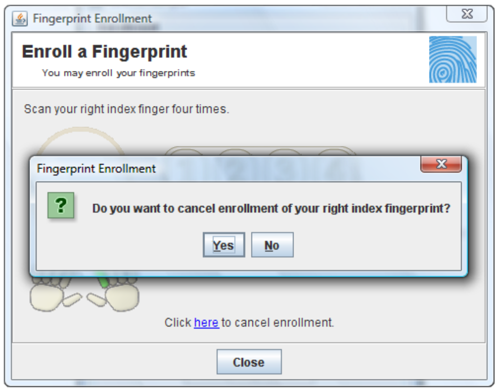
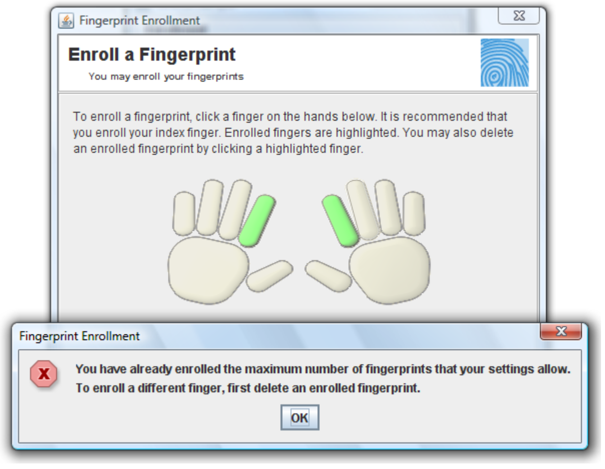
Graphical user interface	**User actions and user interface feedback**
Picture 9  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints To enroll a fingerprint, click a finger on the hands below. It is recommended that you enroll your index finger. Enrolled fingers are highlighted. You may also delete an enrolled fingerprint by clicking a highlighted finger. Close	The hands element indicates that the right index fingerprint is enrolled, that is, the finger is green. The enrolledFingers set now contains a DPFPFingerIndex.RIGHT_INDEX value.
Picture 10  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints Scan your right index finger four times. To begin, sw reader. Repe **Enrollment failed** These scans are not suitable to enroll your fingerprint. To try again, touch the fingerprint reader with your right index finger. Click here to cancel enrollment. Close	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  Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints Scan your right index finger four times. Fingerprint Enrollment **Do you want to cancel enrollment of your right index fingerprint?** Yes No Click here to cancel enrollment. Close	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 1. `DPFPEnrollmentControl` object graphical user interface: Enrolling a fingerprint *(continued)*

Graphical user interface	**User actions and user interface feedback**
Picture 12  Fingerprint Enrollment Enroll a Fingerprint You may enroll your fingerprints To enroll a fingerprint, click a finger on the hands below. It is recommended that you enroll your index finger. Enrolled fingers are highlighted. You may also delete an enrolled fingerprint by clicking a highlighted finger. Fingerprint Enrollment You have already enrolled the maximum number of fingerprints that your settings allow. To enroll a different finger, first delete an enrolled fingerprint. OK	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 2. `DPFPEnrollmentControl` graphical user interface: Unenrolling (deleting) a fingerprint template

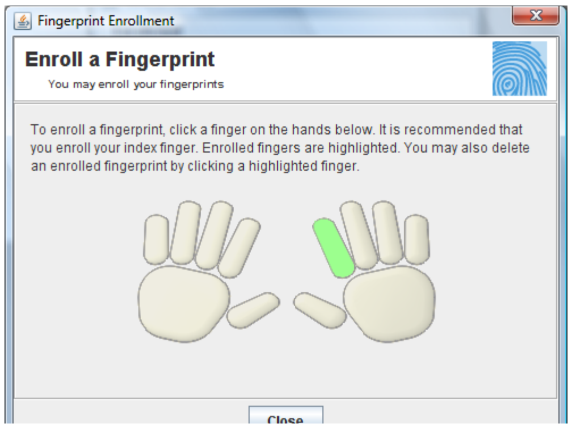
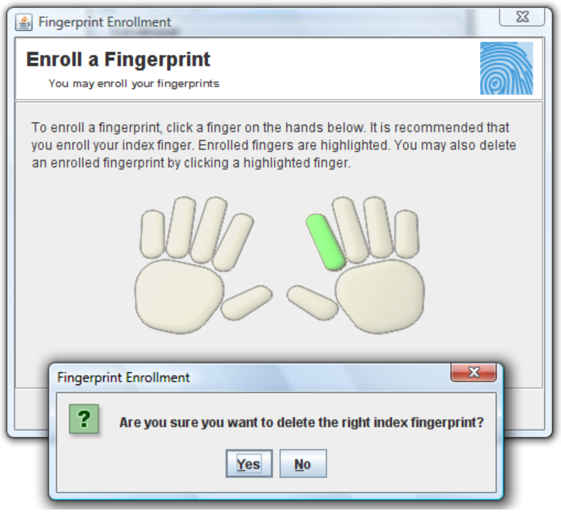
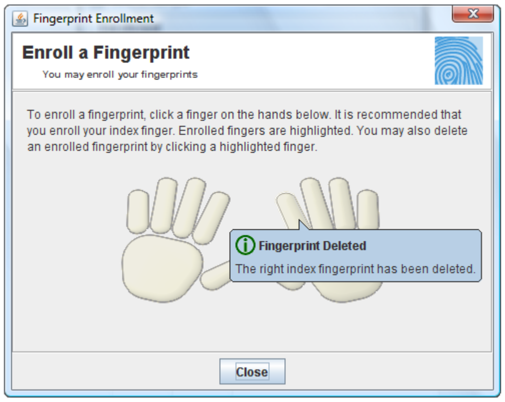
Graphical user interface	User actions and user interface feedback
 Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints To enroll a fingerprint, click a finger on the hands below. It is recommended that you enroll your index finger. Enrolled fingers are highlighted. You may also delete an enrolled fingerprint by clicking a highlighted finger. Close	The hands element indicates that the right index fingerprint is enrolled, that is, the finger is green. The enrolledFingers set now contains a DPFPFingerIndex.RIGHT_INDEX value.
 Fingerprint Enrollment **Enroll a Fingerprint** You may enroll your fingerprints To enroll a fingerprint, click a finger on the hands below. It is recommended that you enroll your index finger. Enrolled fingers are highlighted. You may also delete an enrolled fingerprint by clicking a highlighted finger. Fingerprint Enrollment Are you sure you want to delete the right index fingerprint? Yes No	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).

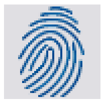
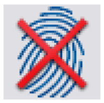
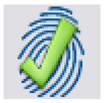
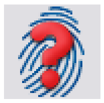
Table 2. `DPFPEnrollmentControl` graphical user interface: Unenrolling (deleting) a fingerprint template

Graphical user interface	**User actions and user interface feedback**
 Fingerprint Enrollment. **Enroll a Fingerprint** You may enroll your fingerprints. To enroll a fingerprint, click a finger on the hands below. It is recommended that you enroll your index finger. Enrolled fingers are highlighted. You may also delete an enrolled fingerprint by clicking a highlighted finger. **Fingerprint Deleted** The right index fingerprint has been deleted. Close	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 enrolledFingers set is now empty. The green color is removed from the right index finger, indicating that the associated fingerprint is no longer enrolled.

DPFPVerificationControl Graphical User Interface

The graphical user interface included with the **`DPFPVerificationControl`** 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 3* illustrates and describes the interaction between the user and the graphical user interface.

Table 3. **`DPFPVerificationControl`** 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 **`captureCompleted`** 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 **`captureCompleted`** event of the verification control event handler is fired and returns a status of **`Failure`** in the **`EventHandlerStatus`** parameter.

Redistribution 7

You may redistribute the files in the RTE\Install and the Redist folders in the One Touch for Windows SDK: Java Edition 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: Java Edition, you need to provide the appropriate 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: Java Edition software package to your end users and to incorporate these files into your application 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

- 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
- DPOTJni.msm

  This merge module contains the following files:
  - dpotjni.jar
  - otdpfpjni.dll
  - otfxini.dll
  - otmcjni.dll
- DPOTJni_x64.msm

  This merge module contains the following files:
  - dpotjni.jar
  - otdpfpjni.dll
  - otfxini.dll
  - otmcjni.dll
- DPOTJava.msm

  This merge module contains the following files:
  - dpfpenrollment.jar
  - dpfpverification.jar
  - dpotapi.jar
- DpPolicies_OTW.msm
  - This merge module contains registry keys only.
- DpProCore.msm

  This merge module contains the following files:
  - Dpdevts.dll
  - Dpsvinfo2.dll
  - Dptsclnt.dll
- DpProCore_x64.msm

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

- Dpsvinfo2.dll
- Dptsclnt.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: Java Edition software package to your end users pursuant to the terms of this section and 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: Java Edition software package. You may copy and redistribute to your end users 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: Java Edition 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.

Developing Citrix-aware applications 8

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

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

- Program Neighborhood
- Program Neighborhood Agent
- Web Client

In order to utilize this support, your application (or the end-user) will need to copy a file to the client computer and register it. The name of the file is DPICACnt.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 of your software package.
2. Copy the file to the folder on the client computer where the Citrix client components are located (i.e. for the Program Neighborhood client it might be the "Program Files\Citrix\ICA Client" folder).
3. Using the regsvr32.exe program, register the DPICACnt.dll library.

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

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

Setting the False Accept Rate 9

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 82*). 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

You can specify the value of the FAR using the setFARRequested method. The following sample code sets the FAR to a value of MEDIUM_SECURITY_FAR.

```
matcher.setFARRequested(DPFPVerification.MEDIUM_SECURITY_FAR);
```

Achieved FAR

The actual value of the FAR achieved for a particular verification operation can be retrieved using the getFalseAcceptRate method of the DPFPVerificationResult interface (*page 83*).

```
DPFPVerification verification =
DPFPGlobal.getVerificationFactory().createVerification(farRequested);
DPFPVerificationResult result = verification.verify(featureSet, template);
int FAR = result.getFalseAcceptRate();
```

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 10

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

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

Platinum SDK Enrollment Template Conversion for Microsoft Visual C++

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

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

```
#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++ Applications *(continued)*

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

    if (error != Er_OK)
    return false;

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

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

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

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

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

    memcpy(goldBlob, dataGold, blobSizeRequired);

    SafeArrayUnaccessData(varValGold.parray);

    return true;

}
```

Platinum SDK Enrollment Template Conversion for Visual Basic 6.0

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

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

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

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

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 DPFPData object. Fingerprint data objects include DPFPSample (represents a fingerprint sample), DPFPFeatureSet (represents a fingerprint feature set), and DPFPTemplate (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