

DigitalPersona, Inc.

One Touch ID SDK

Version 2.2

Developer Guide



digitalPersona.

DigitalPersona, Inc.

© 2008-2009 DigitalPersona, Inc. All Rights Reserved.

All intellectual property rights in the DigitalPersona software, firmware, hardware, and documentation included with or described in this guide are owned by DigitalPersona or its suppliers and are protected by United States copyright laws, other applicable copyright laws, and international treaty provisions. DigitalPersona and its suppliers retain all rights not expressly granted.

DigitalPersona, One Touch, and U.are.U are trademarks of DigitalPersona, Inc., registered in the United States and other countries. Adobe and Adobe Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Microsoft, Access, Visual Basic, Visual C#, .NET, Visual Studio, Windows, Windows Server, and Windows Vista are either trademarks or registered trademarks of Microsoft Corporation in the United States and other countries. All other trademarks are the property of their respective owners.

This DigitalPersona One Touch ID SDK and the software it describes are furnished under license as set forth in the "License Agreement" screen that is shown during the installation process.

Except as permitted by such license or by the terms of this guide, no part of this document may be reproduced, stored, transmitted, and translated, in any form and by any means, without the prior written consent of DigitalPersona. The contents of this guide are furnished for informational use only and are subject to change without notice. Any mention of third-party companies and products is for demonstration purposes only and constitutes neither an endorsement nor a recommendation. DigitalPersona assumes no responsibility with regard to the performance or use of these third-party products. DigitalPersona makes every effort to ensure the accuracy of its documentation and assumes no responsibility or liability for any errors or inaccuracies that may appear in it.

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 about any errors, omissions, or suggestions for future improvements. Please contact us at

TechPubs@digitalpersona.com

or

DigitalPersona, Inc.
720 Bay Road, Suite 100
Redwood City, California 94063
USA
(650) 474-4000
(650) 298-8313 Fax

Table of Contents

1	Introduction	1
	Target Audience	1
	Chapter Overview	1
	Document Conventions	2
	Notational Conventions	2
	Typographical Conventions	2
	Naming Conventions	3
	Additional Resources	3
	Related Documentation	3
	Online Resources	3
	System Requirements	4
	Supported DigitalPersona Hardware Products	4
	Fingerprint Template Compatibility	5
2	Quick Start	6
	Identification Sample application	6
	Adding Fingerprint Templates	7
	Performing Fingerprint Identification	8
	Removing a user	8
	Clearing the User Collection	9
3	Installation	10
	Installing the SDK	10
	Installing the Runtime Environment (RTE)	11
4	Identification Overview	14
	Fingerprint Recognition	14
	Enrollment	14
	Fingerprint verification	14
	Fingerprint identification	15
	Potential applications	16
	Differences between verification & identification	17
	Identification Terminology and Concepts	18
5	Identification API Reference	21
	Common	21
	DPFP.Data	21
	DPFP.Data.FeatureSet	22
	DPFP.Data.Template	22

- Error Component 23
 - DPFP.Error.ErrorCodes 23
 - DPFP.Error.ExceptionFactory 24
 - DPFP.Error.SDKException 24
- ID Component 25
 - DPFP.ID.Candidate 25
 - DPFP.ID.CandidateIDList 25
 - DPFP.ID.CandidateList 25
 - DPFP.ID.FingerPosition 26
 - DPFP.ID.Identification 27
 - DPFP.ID.User 29
 - DPFP.ID.UserCollection 31
 - DPFP.ID.UserID 34
 - DPFP.ID.EventHandler 35
- 6 Redistribution 36**
 - Determining Additional Memory Requirements 36
- 7 Platinum SDK Enrollment Template Conversion 37**
 - Platinum SDK Enrollment Template Conversion for VB 6.0 39
- Glossary 40**
- Index 42**

The identification of fingerprints has a long history in law enforcement and government applications. Businesses are now incorporating fingerprint identification into their security programs. For example, automatic fingerprint identification is used to control access to physical locations or bank accounts or to record employee attendance.

The One Touch® ID SDK continues in the tradition of DigitalPersona SDKs in providing solid, proven tools for developers to incorporate into mission-critical applications. The One Touch ID SDK extends the functionality of the DigitalPersona One Touch for Windows, Gold and Platinum SDKs with the ability to perform high-speed searches on a collection of enrolled fingerprints.

Developers who have created applications using the DigitalPersona One Touch for Windows SDK, Gold SDK or Platinum SDK can improve their offerings with the addition of identification capabilities.

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.

Target Audience

This guide is for developers who have a working knowledge of the C# or VB.NET programming language. In addition, some familiarity with fingerprint recognition and identification concepts and processes is recommended. The DigitalPersona White Paper: Guide to Fingerprint Recognition (Fingerprint Guide.pdf located in the Docs folder in the One Touch ID SDK product package) contains information on these topics.

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; identifies a number of resources that may assist you in using the One Touch ID SDK; lists the minimum system requirements needed to run the One Touch ID SDK; and provides a compatibility matrix of supported DigitalPersona products.

Chapter 2, Quick Start, briefly discusses the One Touch ID SDK identification sample.

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, Identification Overview, introduces One Touch ID identification terminology and concepts, and provides workflow diagrams and explanations of the Identification API components used to perform the identification operations in the workflow.

Chapter 5, *Identification API Reference*, defines the components, attributes enumerations and events that are part of the Identification API.

Chapter 6, *Redistribution*, identifies the files that you may redistribute according to the end-user license agreement provided in the One Touch ID SDK product package and provides guidelines for estimating the amount of memory required for identification sets and for the fingerprint templates you add to them.

Chapter 7, *Platinum SDK Enrollment Template Conversion*, provides sample code for converting Platinum SDK registration templates to Gold format for use with the One Touch ID SDK.

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 reminders, tips, or suggestions that supplement the material included in this guide.

IMPORTANT: Important notations contain significant information about system behavior.

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

Typographical Conventions

The following typographical conventions are used in this guide:

Typeface	Purpose	Example
Courier bold	Used to indicate computer program code	<pre>if (FAILED(SafeArrayAccessData(varVal.parray, (void**)&data))) return false;</pre>
<i>Italics</i>	Used for emphasis or to introduce new terms For developers who are viewing this document online, text in italics may also indicate hypertext links to other areas in this guide.	A <i>candidate</i> is an enrollee whose fingerprint template(s) is determined to be similar to the fingerprint feature set(s).
Bold	Used in running text for keystrokes and window and dialog box elements	Press Enter . Click the Info tab.

Naming Conventions

The *DPFP.ID* namespace used in Identification API components, functions methods, type definitions, and constants stands for *DigitalPersona FingerPrint*.

Additional Resources

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

Related Documentation

For information about	See
Fingerprint recognition, including the history and basics of fingerprint identification and the advantages of the DigitalPersona's Fingerprint Recognition Algorithm	The DigitalPersona White Paper: Guide to Fingerprint Recognition (Fingerprint Guide.pdf located in the Docs folder in the One Touch ID SDK product package)
Late-breaking news about the product	The Readme.txt files provided in the root directory of the product package as well as in some subdirectories
How to use the DigitalPersona Gold SDK to create registration features (fingerprint templates)	The Gold SDK Developer Guide included with the product
How to use the DigitalPersona One Touch SDKs to create registration features (fingerprint templates)	The One Touch for Windows and One Touch for Linux SDK Developer Guides included with each product package
How to use the DigitalPersona Platinum SDK to convert Platinum fingerprint templates to Gold format	The Platinum SDK Developer Guide included with the product. Sample code is also provided in Appendix 7 on page 37.

Online Resources

To locate the	Go to
DigitalPersona Developer Connection Forum for DigitalPersona Developers	http://www.digitalpersona.com/webforums/
Latest updates for DigitalPersona software products	http://www.digitalpersona.com/support/downloads/software.php
Standards for mapping fingers to finger number positions	The ISO IEC 19794-2 2005 standard at http://www.iso.org/iso/enCatalogueDetailPage.CatalogueDetail?CSNUMBER=38746 or the ANSI INCITS 378-2004 standard at http://webstore.ansi.org/ansidocstore/product.asp?sku=ANSI+INCITS+378-2004

System Requirements

This section lists the minimum software and hardware requirements needed to run the One Touch ID SDK.

- X86-based processor or better
- CD-ROM drive
- Microsoft® Windows® XP (32 and 64 bit) or Microsoft® Windows Vista® (32 and 64 bit)
- Microsoft .NET framework v2.0 or later.
- Administrative privileges
- The minimum amount of RAM required to run your Microsoft Windows operating system. (See also *Determining Additional Memory Requirements on page 36.*)
- Approximate available hard-disk space as follows

One Touch ID SDK	30 MB
One Touch ID Runtime Environment	5 MB
One Touch for Windows Runtime Environment (Always installed with OTID, but optional as part of OTID RTE install.)	15 MB

- USB connector on the computer where the fingerprint reader is to be connected
- Microsoft® Visual Studio® 2005 or later

NOTE: In order to acquire images and create fingerprint templates, you will also need one of the following SDKs installed:

- DigitalPersona Gold SDK
- DigitalPersona Platinum SDK
- DigitalPersona One Touch for Windows SDK 1.4.0 or later

The process of creating registration features/fingerprint templates is outside the scope of this guide and is covered in your One Touch for Windows, Gold or Platinum SDK documentation. Platinum fingerprint templates must be converted to Gold SDK format. Platinum SDK developers can use the sample code on *page 37* to convert your Platinum SDK fingerprint templates to Gold format.

Supported DigitalPersona Hardware Products

The One Touch ID SDK 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

An ✓ in the Template Type column of the following table indicates that the template format for the associated product is compatible with the One Touch ID SDK.

Product	BTF ¹	XTF ²
Gold SDK registration features (default key)	✓	✓
Gold SDK registration features (custom key)	✓	✓
Platinum SDK registration templates converted to compatible format ³	✓	✓
Gold CE SDK registration features	✓	✓
DigitalPersona Pro templates	X	X

1. Basic Template Format
2. Extended Template Format
3. Platinum SDK users can use the sample code on *page 37* to convert your Platinum SDK enrollment templates to compatible format.

The following sample application is included with the One Touch ID SDK and demonstrates the use of the Identification API functions.

NOTE: The process of acquiring fingerprint feature sets¹ and of generating fingerprint templates² is outside the scope of this guide and is covered in your One Touch for Windows, Gold or Platinum SDK documentation. In addition, the One Touch ID SDK does not provide the functionality to retrieve or to store fingerprint templates. You must use your own methods for performing these tasks.

Identification Sample application

The Identification Sample application, available in both C# and VB.NET versions, is designed to introduce you to the fingerprint identification process.

Project files for the two sample applications are CSharp\IdentificationSample VC.vcproj and VBNET\IdentificationSample VB.vbproj. The executable binaries are prebuilt and supplied within each project's Release folder.

The sample application is an implementation example for Kiosk-based identification. Capture and processing (OTW), and identification (OTID) all take place on the same computer. Identification may instead be done remotely and independent of fingerprint capture.

This section takes you through the steps of:

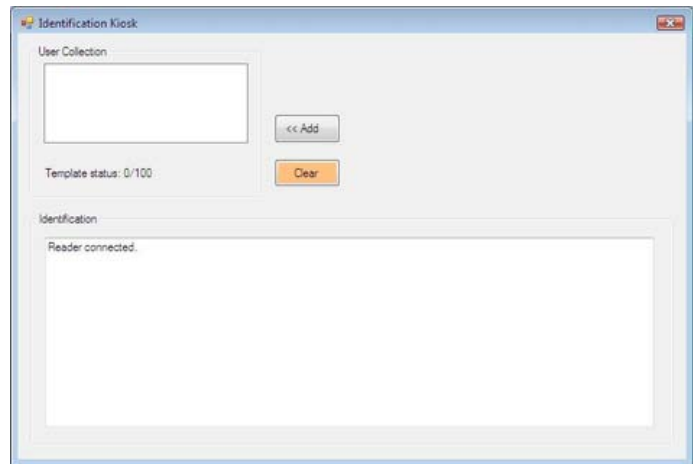
1. Adding Fingerprint Templates
2. Performing Fingerprint Identification
3. Removing a user
4. Clearing the User Collection

1. A *fingerprint feature set* is the output of a completed fingerprint feature extraction process applied to a fingerprint sample. Feature set(s) can be produced for the purpose of fingerprint enrollment or for the purpose of fingerprint recognition.

2. A *fingerprint template* is 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).

To use the Identification Sample application

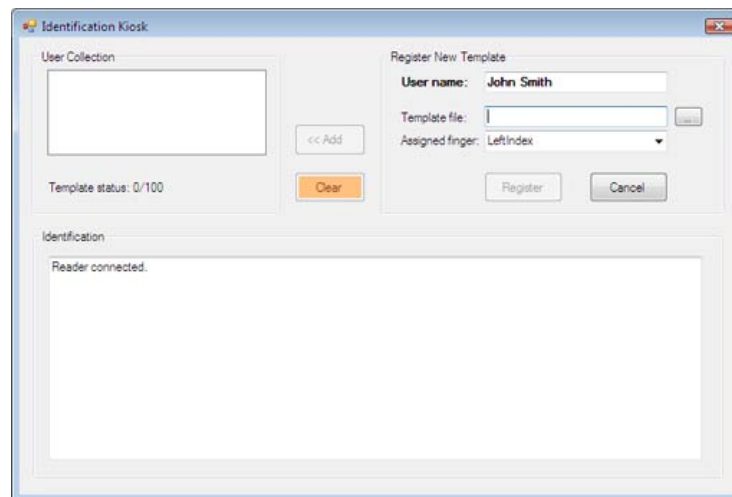
1. Within the directory where you installed the One Touch ID SDK, locate the Samples folder and the subfolder for the language you will be using, i.e. C# or VB.NET.
2. Click on **IdentificationSample CS.exe** (for C#) or **IdentificationSample VB.exe** for VB.NET.
3. The Identification Sample window displays.



Adding Fingerprint Templates

This exercise will illustrate the use of the UserCollection object, and how to associate fingerprint templates to its members. Identification of a user is accomplished by matching a user's fingerprint template against those in the UserCollection object.

1. Click **Add**.
2. The **Enroll New Template** panel displays in the upper right of the window.



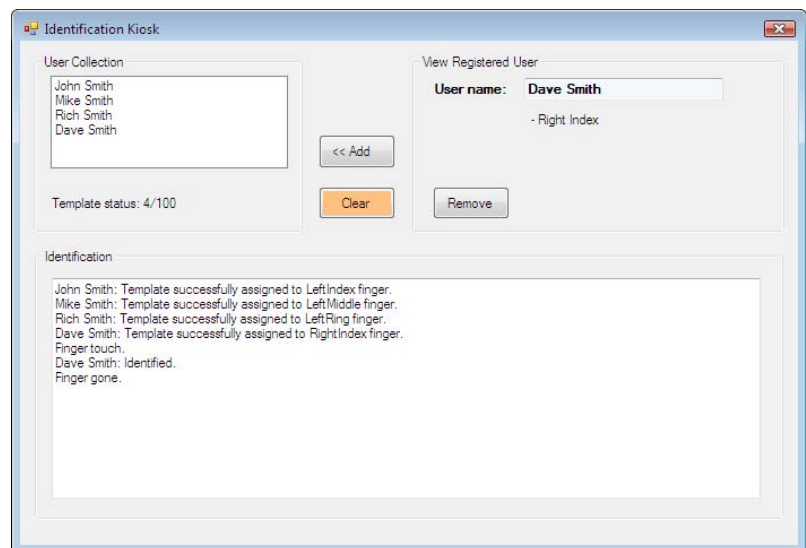
3. Enter a user name in the **User name** field.
4. In the **Template File** field, enter the path to the user's fingerprint template, or click the ellipsis (...) button to browse for the fingerprint template.
5. In the **Assigned Finger** field, select a finger description that defines which finger was used for the fingerprint template.
6. Click **Enroll**.

NOTE: During enrollment, the program will perform a check to determine whether or not this template has already been enrolled. If it has, a message will be logged, "Please note: This template is already registered."

Performing Fingerprint Identification

In this exercise, you will perform fingerprint identification.

1. Touch or swipe the fingerprint reader.
2. If your fingerprint matches one of the fingerprint templates in the set, you are identified and your name is returned in the Identification field.



Removing a user

Removing a user deletes all fingerprint templates for that user from the fingerprint set. In this exercise, you will remove a user from the user collection.

1. In the **User Collection** area, select the user to be removed.
2. Click **Remove**.

3. The user and their fingerprint templates are removed from the user collection.

Clearing the User Collection

In this exercise, you will clear all users from the User Collection.

1. Click **Clear**.
2. All users are deleted from the User Collection.

This chapter contains instructions for installing the various components of the One Touch ID SDK, identifies the files and folders that are installed on your hard disk, and describes the contents of the One Touch ID SDK product package.

The One Touch ID SDK will not install on a computer where DigitalPersona Password Manager is installed.

Installing the SDK

If you are a developer who wants to create applications based on the One Touch ID Identification API functions, follow the instructions in this section to install the One Touch ID SDK.

To install the One Touch ID SDK

1. Download the One Touch ID SDK product package to your local drive.
2. Unzip the product package.
3. In the SDK folder, double-click **Setup.exe**.
4. Follow the installation instructions as they appear.

Table 1 describes the files that are packaged within the MSI installation package, and installed on your hard disk.

Table 1. One Touch ID SDK files installed

File	Description
All RTE files listed in Table 2 plus	
<INSTALLDIR>\DigitalPersona\OneTouch I.D. SDK\.NET\Docs	.NET Documentation
<INSTALLDIR>\DigitalPersona\OneTouch I.D. SDK\.NET\Bin	.NET binary files
<INSTALLDIR>\DigitalPersona\OneTouch I.D. SDK\.NET\Samples	.NET Samples

Installing the Runtime Environment (RTE)

The One Touch ID RTE is provided for distribution/deployment with the application that you create. It must exist on the computer that runs your application created with the One Touch ID SDK. The RTE is generally included in your application installer.

To install the One Touch ID Runtime Environment

1. In the RTE folder, double-click **Setup.exe**.
2. Follow the installation instructions as they appear.

Table 2 identifies the RTE files that are installed.

NOTE: Files that are in bold in the list below are always installed. Non-bold files are part of the optional One Touch for Windows RTE.

Table 2. One Touch ID RTE installed files (32 bit)

Installation Path	Filename
<SystemFolder>	DpIdent.dll
	DPFPApi.dll
	DpClback.dll
	dpHFtrEx.dll
	dpHMatch.dll
	DPFpUI.dll
<INSTALLDIR>\DigitalPersona\Bin	DPCOper2.dll
	DPDevice2.dll
	DPDevTS.dll
	DpHostW.exe
	DPmsg.dll
	DPMux.dll
	DpSvInfo2.dll
	DPTSCInt.dll
	DPCrStor.dll
<INSTALLDIR>\DigitalPersona\Bin\COM-ActiveX	DPFPShrX.dll
	DPFPEngX.dll

Table 2. One Touch ID RTE installed files (32 bit)

Installation Path	Filename
	DPFPDevX.dll
	DPFPCtIX.dll
[GlobalAssemblyCache]	DPFPO2MNET.dll
	DPFPShrNET.dll
	DPFPDevNET.dll
	DPFPEngNET.dll
	DPFPVerNET.dll
	DPFPGuiNET.dll
	DPFPCtIXTypeLibNET.dll
	DPFPCtIXWrapperNET.dll
	DPFPShrXTypeLibNET.dll

Table 3. One Touch ID RTE installed files (64 bit)

Installation Path	Filename
<System64Folder>	DpIdent.dll
	DPFPApi.dll
	DpClback.dll
	dpHFtrEx.dll
	dpHMatch.dll
	DPFpUI.dll
<SystemFolder>	DpIdent.dll
	DPFPApi.dll
	DpClback.dll
	dpHFtrEx.dll
	dpHMatch.dll
	DPFpUI.dll

Table 3. One Touch ID RTE installed files (64 bit)

Installation Path	Filename
<INSTALLDIR>\DigitalPersona\Bin	DPTSCInt.dll
<Program Files (x86)>\DigitalPersona\Bin	DPCOper2.dll
	DPDevice2.dll
	DPDevTS.dll
	DpHostW.exe
	DPmsg.dll
	DPMux.dll
	DpSvInfo2.dll
	DPCrStor.dll
<INSTALLDIR>\DigitalPersona\Bin\COM-ActiveX	DPFPShrX.dll
	DPFPEngX.dll
	DPFPDevX.dll
	DPFPctIX.dll
<Program Files (x86)>\DigitalPersona\Bin\COM-ActiveX	DPFPShrX.dll
	DPFPEngX.dll
	DPFPDevX.dll
	DPFPctIX.dll
[GlobalAssemblyCache]	DPFPO2MNET.dll
	DPFPShrNET.dll
	DPFPDevNET.dll
	DPFPEngNET.dll
	DPFPVerNET.dll
	DPFPGuiNET.dll
	DPFPctIXTypeLibNET.dll
	DPFPctIXWrapperNET.dll
	DPFPShrXTypeLibNET.dll

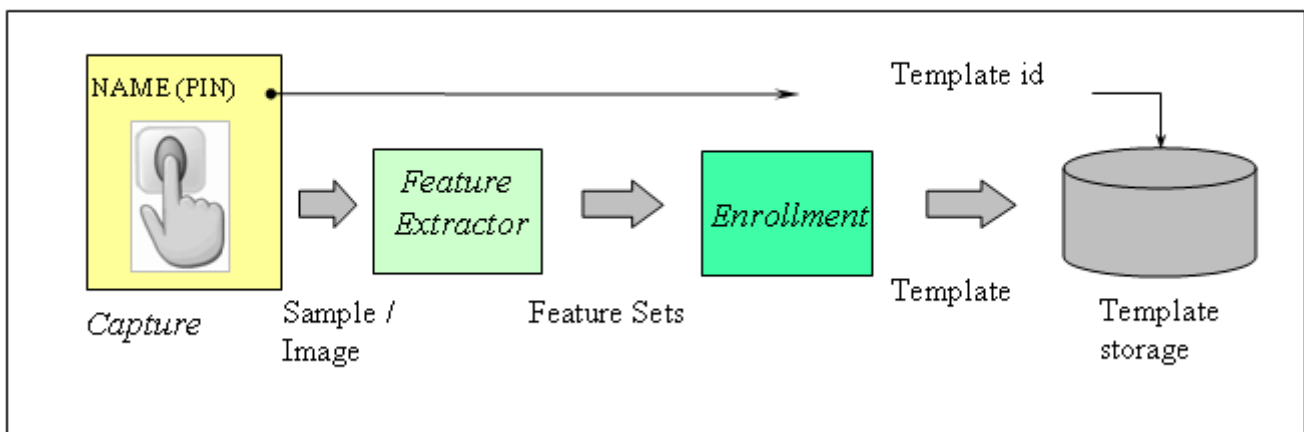
This chapter provides a brief overview of Fingerprint Recognition, introducing the concepts of enrollment, verification and identification. It also introduces One Touch ID identification terminology and concepts, puts identification operations in context by providing three fictional scenarios, and includes corresponding workflow diagrams and explanations of the Identification API functions used to perform the identification operations in the workflow.

Fingerprint Recognition

Fingerprint recognition consists of three identifiable tasks: enrollment, verification and identification.

Enrollment

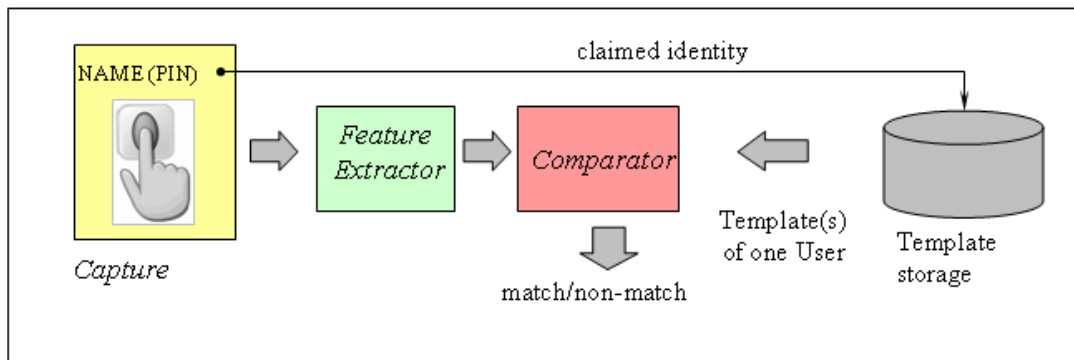
The enrollment process is common to, and precedes, all other fingerprint recognition tasks. In any fingerprint recognition application, one or more fingers are enrolled for each user. The enrollment process for a finger results in the production of an enrollment template, which is then placed into a template storage area.



Fingerprint verification

In a fingerprint verification system, the user makes a positive claim of identity to the system (for example, "I am Joe."). The system then retrieves from template storage the templates that correspond to the claimed identity

and conducts a one-to-one comparison between the recognition feature set and the template(s) resulting in a "match" or "non-match" decision.



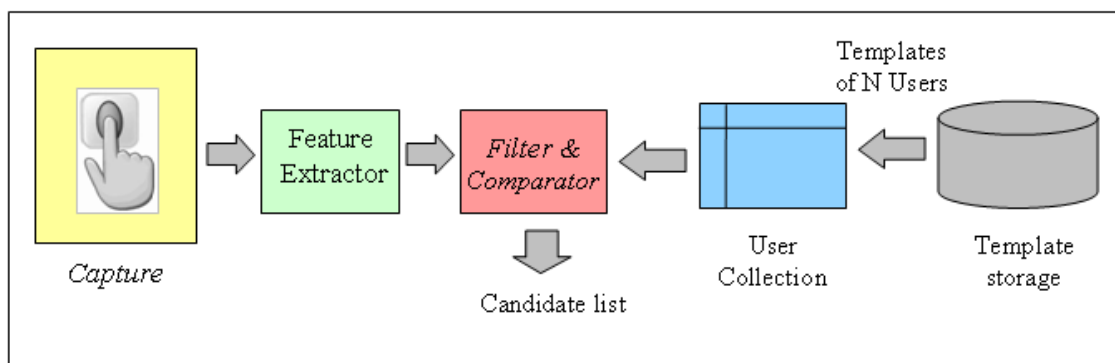
The claim in a verification system is always "positive", that is, it is in the interest of the user to receive a "match" decision from the fingerprint verification system. A positive claim of identity may be presented by giving a username, login id, user id, token such as smart card, or by other identifying means and is used in retrieving the corresponding templates from the template storage.

Response time, recognition reliability, memory consumption of a verification system are not affected by the number of templates present in the template storage.

For development of fingerprint verification applications, use the One Touch for Windows SDK.

Fingerprint identification

In a fingerprint identification system, the application will first add one or more (or all) of the enrollment templates into a memory-resident User Collection. The primary use of the User Collection is to lower the response time for identification. Response time, recognition reliability, and memory consumption in an identification system depend on the number of Users and the number of fingers in the User Collection.



In using an identification system, the User may make a positive claim of presence (that is, "I am present in the identification set."), or a negative claim of presence (that is, "I am not present in the identification set.").

- In a positive claim of presence, it is in the interest of the User to receive a positive answer from the fingerprint identification system. For example, “You have been found in the identification set as Joe.”
- In a negative claim of presence, it is in the interest of the User to receive a negative answer from the fingerprint identification system. For example, “You were not found in the identification set.”).

The fingerprint identification system relays its decision through the Candidate List.

Potential applications

Whether it is in the interest of a User to make a positive claim of an identity, or a positive or negative claim of presence, depends on the purpose of the application. The application context also determines how the results (that is, the Candidate List), are used.

1. A bank employee uses a fingerprint recognition system to login to her corporate domain server running the Windows operating system. In this application, she makes a positive claim of identity (for example, “I am Mary.”) by typing her username on the keyboard and submitting her fingerprint. This application uses verification only, not identification, and is easily scalable since the number of enrolled bank employees does not impact response time, recognition reliability, or memory usage.
2. A pharmacist uses a fingerprint recognition system to electronically sign each bottle of medicine dispensed. Even though he may have a username for his account on the local computer or domain, he may wish to simply make a positive claim of presence (e.g., “I am a pharmacist present in the User Collection.”) by submitting his fingerprint – thus relieving him from the inconvenience of typing a username on the keyboard each time. When there is a need to use the system frequently, the convenience of not typing a username can really add up.

This application would use fingerprint identification. The results are returned in the Candidate List, which will be empty or generally consist of one element containing the identifier, such as userID, of the identified User.

In two rare cases, the Candidate List may contain more than one element:

- If a single User is present multiple times in the User Collection. A user can be present only once under the same name, but could theoretically be present multiple times under different identities, i.e. user id in the API.
- A false positive identification error.

Since these are rare occurrences, the application may generally look for only zero or one element in the Candidate List and ignore the rest of the elements.

3. A bank wants to prevent certain people from opening new accounts: for example, customers who have previously defrauded the bank. The bank would use fingerprint identification for this purpose and operate it at a high false positive identification rate (that is, a low false negative identification rate).

The application will create a new User Collection and add the enrollment templates of the unwanted Users into it. When a new account applicant provides their fingerprint for identification, they are essentially making a negative claim of presence (that they are not present in the “unwanted” User Collection. The fingerprint identification application will return an empty Candidate List if no match is found. The Candidate List will contain one element with the id of the User, if one match is found. In case

the User is present in the identification set under more than one identities, the Candidate List will have two or more elements containing a list of all the matching UserIDs. If a very high false negative identification rate has been used, the Candidate List may contain a few elements corresponding to those templates in the User Collection that appeared to be similar to the recognition feature set.

Differences between verification & identification

In verification, the User makes an identity claim and the templates corresponding to that identity are retrieved from the template storage for comparison with the recognition feature set. However, in identification, a memory-resident User Collection must first be prepared. Addition of templates into the User Collection takes a bit of time, but needs to be performed only at start up time and when a new enrollment template has become available. It takes about 8 seconds to add 2,000 templates into the User Collection.

Verification returns a “match or “non-match” decision, while identification returns a Candidate List, which in most cases will either be empty or have one element. In some applications (such as example 3 above), it may be useful for the application to examine if there is more than one element in the Candidate List. If there is more than one element in the candidate list, the application needs to make the final decision. The application may ask for another finger or not grant access (in an access control application) or may alert a banker for manual resolution, etc.

Verification response time, recognition reliability, and memory usage do not depend on the number of templates in the template storage. Identification response time, recognition reliability, and memory usage depend upon the number of Users and number of templates in the User Collection.

- Response time - As the size of the User Collection increases, an identification operation will take longer to complete. For example, an identification operation that takes typically under 0.2 second on a Pentium4 machine when the User Collection contains 2,000 templates will take up to one second when the User Collection contains 20,000 templates.
- Recognition reliability - As the number of templates in a User Collection increases, the chance of a false positive identification error increases. However, since the false positive identification rate is preset, it is internally adjusted with every increase in the number of templates in the User Collection. Consequently, the false negative identification rate somewhat increases with the increase in the User Collection. For example, increase of User Collection from 500 templates to 20,000 templates may double the false negative identification rate.
- Memory usage - As the size of the User Collection increases, it requires more memory (RAM). In the current implementation, the actual creation of the User Collection requires about 12MB of RAM, plus the addition of each template into the User Collection requires about 40KB. So, a User Collection of 20,000 templates will require roughly 800MB of RAM.

Due to the dependence of identification's response time, recognition reliability, and memory usage on the number of templates (N) in the User Collection, it is advisable to keep N as small as possible. For example, in some applications, geographic location may be used to segment the template storage such that each individual User Collection is smaller. If this is not possible, and the User Collection size is large, multiple computers or processors may be used to lower the response time and spread the memory usage, and multiple fingers may be used to get better recognition reliability.

Such scalability considerations are application specific and need to be considered by the application developer as part of their design.

Identification Terminology and Concepts

This section defines the One Touch ID identification terminology and concepts used throughout this guide.

Identification Operation

An *identification operation* is a sequence of function calls that performs identification tasks. The Identification One Touch ID components enable you to perform fingerprint identification.

Fingerprint identification compares the fingerprint feature set(s)¹ from one or more fingers of one user with the fingerprint templates² of more than one enrollee³, makes a decision, and produces a candidate list.

Users and User Collections

An Identification procedure requires the use of a User Collection. User Collections aggregate Users, with each User identifiable by a unique UserID. A User contains a pool of fingerprint templates, each associated with a specific finger.

User Collections are capped by template capacity, set on constructions. This capacity spreads across all of the collection members. A User may belong to several collections.

WARNING: Users and User Collections are stored in memory. You should not rely on such volatile data structures as permanent template storage.

The same User Collection can be used in multiple identification operations at the same time. A User Collection cannot be modified (users added to or removed from it) or destroyed while there are any unfinished identification operations running.

Identification

Identification operations are performed by the identification component over a specified User collection. A successful identification operation will result in a list of identification candidate IDs, satisfying a preset false positive identification rate.

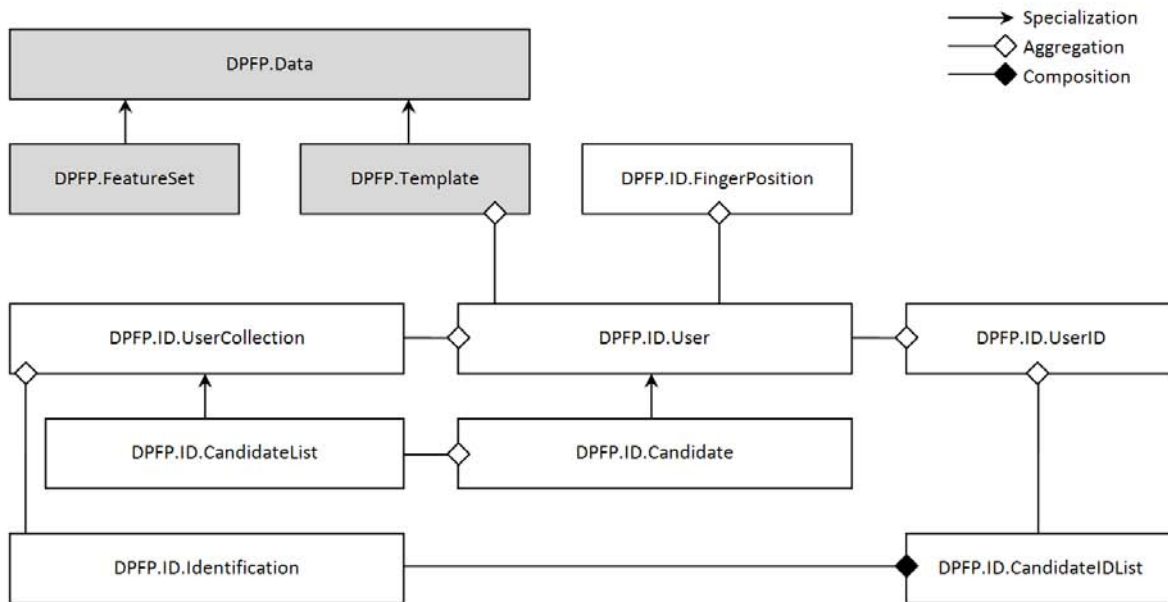
As the user collection's template count increases, an identification operation may take longer to complete. The identification component manages asynchronous events, to be received by a registerable identification event

1. A *fingerprint feature set* is the output of a completed fingerprint feature extraction process applied to a fingerprint sample. Feature set(s) can be produced for the purpose of fingerprint enrollment or for the purpose of fingerprint recognition.
2. A *fingerprint template* is 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).
3. The user whose fingerprint template(s) are held in the fingerprint data storage subsystem.

handler. These events include status and identify messages. Both event interfaces allow event handlers to terminate an ongoing identification operation.

A Candidate List is a specialization of User collection. It receives a Candidates ID list on construction, and builds a User Collection based on those specific IDs. Candidate Lists can be fed to a new Identification operation, when the need for a multi-finger identification arises.

Component Interaction



DPFP.DATA and its specializations, DPFP.Template and DPFP.FeatureSet are part of the One Touch for Windows SDK (.NET version). They seamlessly integrate into One Touch ID.

All ID-related components are located within the DPFP.ID namespace.

DPFP.ID.User aggregates templates assigned to finger positions (defined in DPFP.ID.FingerPosition). It provides the interface for clearing, assignment, removal, retrieval, and checking on assigned templates. Each DPFP.ID.User is identified by a unique ID (DPFP.ID.UserID), provided upon construction.

DPFP.ID.UserCollection aggregates DPFP.ID.User objects. It provides the interface for clearing, adding, removing, retrieving, and checking on collection members. User collections have a preset template capacity, provided on construction. DPFP.ID.UserCollection also provides the interface to report template and member count statuses. A DPFP.ID.User may belong to several DPFP.ID.UserCollection objects.

DPFP.ID.Candidate extends DPFP.ID.User by associating it to a specified Identification procedure. A DPFP.ID.Candidate is a member of DPFP.ID.CandidateList. DPFP.ID.CandidateList objects are created by supplying a DPFP.ID.CandidateIDList, which is a product of an identification procedure.

DPFP.ID.Identification objects provide for an identification mechanism over a predefined DPFP.ID.UserCollection. Identification objects allow to identify a single or a pair of DPFP.FeatureSet objects (obtained elsewhere), as well as to set a custom False Positive Identification Rate (see definition elsewhere), if required. Identifications result in a DPFP.ID.CandidateIDList object, which can be used in turn to create a DPFP.ID.CandidateList.

DPFP.ID.Identification peruses an optional EventHandler interface, DPFP.ID.EventHandler. Objects implementing this interface can intercept events tied to an ongoing identification procedure. These events include progress, status and successful identifications, and should be employed asynchronously.

This chapter defines the functions, data structures, and type definitions that are part of the Identification API.

Common

DPFP.Data

Type: Abstract class, Data container

Description: A base abstract class for the data that is common to all fingerprint data objects.

Operations

Method	Description	Parameters	Exceptions
Data() void	Default constructor		
Data() void	Default constructor	Stream [in] DataStream Constructs data from a given stream	
Serialize() void	Serializes a data object and returns it as an array of bytes	byte[] [inout] ArrayOfBytes Array of bytes	Error.ErrorCodes.Internal Failed to serialize.
Serialize() void	Serializes a data object to a stream	Stream [in] DataStream Data Stream	Error.ErrorCodes.Internal Failed to serialize.
DeSerialize() void	Deserializes a data object returned by the Serialize method	byte[] [in] ArrayOfBytes Array of bytes	Error.ErrorCodes.Internal Failed to deserialize.
DeSerialize() void	Deserializes a data object from a stream	Stream [in] DataStream	Error.ErrorCodes.Internal Failed to deserialize.
Bytes() byte	Returns embedded raw data.		
Size() uint	Returns embedded raw data size in bytes		

DPFP.Data.FeatureSet

Type: Class

Description: Represents a fingerprint feature set.

Operations

Method	Description	Parameters	Exceptions
FeatureSet() void	Default constructor		
FeatureSet() void	Data stream to deserialize	Stream [in] <code>DataStream</code> Data size in bytes	<code>Error.ErrorCodes.Internal</code> Failed to deserialize.

DPFP.Data.Template

Type: Class

Description: Represents a fingerprint template.

Operations

Method	Description	Parameters	Exceptions
FeatureSet() void	Default constructor		
FeatureSet() void	Data stream to deserialize	Stream [in] <code>DataStream</code> Data size in bytes	<code>Error.ErrorCodes.Internal</code> Failed to deserialize.

Error Component

DPFP.Error.ErrorCodes

Type: Enumeration

Description: Error codes for SDK exceptions.

Attributes

Attribute	Description
Success	Operation completed successfully.
NotInitialized	Some Engine components are missing or inaccessible.
InvalidParameter	One or more parameters are not valid.
NotImplemented	Feature is not implemented.
IO	A generic I/O file error occurred.
Not Found	Not found.
IndexOccupied	Index position is occupied.
NoMemory	There is not enough memory to perform the action.
Internal	An unknown internal error occurred.
BadSetting	Initialization settings are corrupted.
UnknownDevice	The requested device is not known.
InvalidBuffer	A buffer is not valid.
FeatureSetTooShort	The specified fingerprint feature set or fingerprint template buffer size is too small.
InvalidContext	The given context is not valid.
InvalidFeatureSetType	The feature set purpose is not valid.
InvalidTemplate	Supplied template is invalid.
InvalidFeatureSet	Decrypted fingerprint features are not valid. Decryption may have failed.
Unknown	An unknown exception occurred.

DPFP.Error.ExceptionFactory

Type: Class

Description: Generates SDK specific exceptions.

Operations

Method	Description	Parameters
Static Throw() void	Generates exceptions based on specified return code	ErrorCodes [in] ErrorCode Source return code string [in] ErrorMessage Error Message
Static Throw() void	Generates an internal application exception	Exception [in] Exception Embedded Exception ErrorCodes [in] ErrorCode Error Code string [in] ErrorMessage Error Message
Static Throw() void	Generates an internal application exception	Exception [in] Exception Embedded Exception string [in] ErrorMessage Error Message

DPFP.Error.SDKException

Type: Specializes ApplicationException

Description: OTID SDK Exception container.

Operations

Method	Description	Parameters
SDKException()	Initialize with Error code and Error Message	Exception [in] Exception ErrorCodes [in] ErrorCode Error Code string [in] ErrorMessage Error Message
ErrorCode() ErrorCodes	Returns embedded Error Code	

ID Component

DPFP.ID.Candidate

Type: Class specializes DPFP.ID.User.

Description: A User identified as a candidate through an identification operation.

Operations

Method	Description	Parameters
Identification() Identification	Returns the identification associated with this candidate	

DPFP.ID.CandidateIDList

Type: Class specializing List<UserID>.

Description: A list of Candidate IDs obtained through identification.

Operations

Method	Description	Parameters
Identification() Identification	Returns the identification associated with this candidate ID list	

DPFP.ID.CandidateList

Type: Specializes DPFP.ID.UserCollection.

Description: A list of candidates obtained through identification. A developer may choose to create CandidateLists in order to retrieve verbose identification information per candidate, or perform secondary (or tertiary) fingerprint operations.

Operations

Method	Desc	Parameters	Exceptions
CandidateList() void	Constructs a candidate list		
		CandidateIDList [in] CandidateIDList	DPFP.Error.ErrorCodes.InvalidParameter Invalid user collection attached to supplied Candidate ID List.
Identification() Identification	Returns the identification associated with candidate list		

DPFP.ID.FingerPosition

Type: Enumeration

Description: Template finger position.

Attributes

Attribute	Description
Unidentified	Finger position is unidentified
RightThumb	Right thumb
RightIndex	Right index finger
RightMiddle	Right middle finger
RightRing	Right ring finger
RightLittle	Right little finger
LeftThumb	Left thumb
LeftIndex	Left index finger
LeftMiddle	Left middle finger
LeftRing	Left ring finger
LeftLittle	Left little finger

DPFP.ID.Identification

Type: Class

Description: One to Many identification component.

Operations

Method	Desc	Parameters	Exceptions
Identification() void	Creates a new Identification operation over a specified UserCollection, with a specified FPIR.		
		UserCollection [in] UserCollection uint [in] FalsePositiveIdentificationRate False positive identification rate	DPFP.Error.ErrorCodes.InvalidParameter Invalid supplied user collection DPFP.Error.ErrorCodes.Internal Failed to initialize identification operation DPFP.Error.ErrorCodes.InvalidContext Failed to create identification context
Identification() void	Creates a new Identification operation over a specified UserCollection		
		UserCollection [in] UserCollection	DPFP.Error.ErrorCodes.InvalidParameter Invalid supplied user collection DPFP.Error.ErrorCodes.Internal Failed to initialize identification operation DPFP.Error.ErrorCodes.InvalidContext Failed to create identification context
Identify() CandidateIDList	Fetches a candidate list (ID.CandidateIDList) identifying supplied feature set. If no candidates are found, the returned list shall be empty. Since this blocking operation may take a while to complete over large user collections, an asynchronous event system is being employed to relay information to an event handler. Please see DPFP.ID.EventHandler and the EventHandler property of Identification for further information.		

Method	Desc	Parameters	Exceptions
		FeatureSet [in] FeatureSet	<p>DPFP.Error.ErrorCodes.Internal Identification failure: out of memory or failure to terminate properly.</p> <p>DPFP.Error.ErrorCodes.InvalidContext Invalid identification context: an identification operation with the same context is currently running. Please wait for it to complete.</p>
Identify() CandidateIDList	<p>Fetches a candidate list (ID.CandidateIDList) identifying users matching both supplied feature sets.</p> <p>If no candidates are found, the returned list shall be empty.</p> <p>Since this blocking operation may take a while to complete over large user collections, an asynchronous event system is being employed to relay information to an event handler. Please see DPFP.ID.EventHandler and the EventHandler property of Identification for further information.</p>		
		<p>FeatureSet [in] FirstFeatureSet</p> <p>FeatureSet [in] SecondFeatureSetFeatureSet</p>	<p>DPFP.Error.ErrorCodes.Internal Identification failure: out of memory or failure to terminate properly.</p> <p>DPFP.Error.ErrorCodes.InvalidContext Invalid identification context: an identification operation with the same context is currently running. Please wait for it to complete.</p>
Identify() CandidateIDList	<p>Fetches a candidate list (ID.CandidateIDList) identifying users matching the supplied template.</p> <p>If no candidates are found, the returned list shall be empty.</p> <p>Since this blocking operation may take a while to complete over large user collections, an asynchronous event system is being employed to relay information to an event handler. Please see DPFP.ID.EventHandler and the EventHandler property of Identification for further information.</p>		

Method	Desc	Parameters	Exceptions
		Template [in] Template	<p>DPFP.Error.ErrorCodes.Internal Identification failure: out of memory or failure to terminate properly.</p> <p>DPFP.Error.ErrorCodes.InvalidContext Invalid identification context: an identification operation with the same context is currently running. Please wait for it to complete.</p>
EventHandler() void	Loads an identification event handler. Set this property to null to clear all registered event handlers. By default, no event handlers are registered.		
		EventHandler [in] EventHandler	
FalsePositiveIdentificationRate() uint	Returns and sets the threshold on the false-positive identification rate		

DPFP.ID.User

Type: Class

Description: An object providing for User-related template manipulations.

Operations

Method	Desc	Parameters	Exceptions
User() void	Constructs a user identified by a supplied unique ID		
		UserID [in] ID	<p>DPFP.Error.ErrorCodes.InvalidParameter ID supplied is invalid.</p> <p>DPFP.Error.ErrorCodes.Internal ID supplied is invalid or corrupt.</p>
User() void	Constructs a user identified by a supplied unique string		
		String [in] ID	
User() void	Copy constructs a user (User ID is duplicated.)		
		User [in] User	
AddTemplate() void	Adds a Template associated with a FingerPosition		

Method	Desc	Parameters	Exceptions
		Template [in] Template FingerPosition [in] FingerPosition	DPFP.Error.ErrorCodes.InvalidParameter Template supplied is invalid, empty, or corrupt. DPFP.Error.ErrorCodes.IndexOccupied Supplied FingerPosition is already associated with another template. DPFP.Error.ErrorCodes.NoMemory Out of memory or maximum template capacity reached.
RemoveTemplate() void	Removes a Template associated with a given FingerPosition		
		FingerPosition [in] FingerPosition	DPFP.Error.ErrorCodes.NotFound Supplied FingerPosition doesn't contain a template to remove. DPFP.DPFP.Error.ErrorCodes.Internal Failed to remove template.
TemplateExists() bool	Checks whether User has a template already associated with a specified FingerPosition		
		FingerPosition [in] FingerPosition	
Template() void	Returns a template associated with a finger		
		FingerPosition [in] Finger	
ClearTemplates() void	Clears all templates associated to User		
ID() UserID	Returns unique ID identifying User		
FingerPositions() List<FingerPosition>	Returns enrolled finger positions for User		

DPFP.ID.UserCollection

Type: Class implementing IDisposable.

Description: An object providing for User collection management. DPFP.ID.UserCollection and its respective DPFP.ID.User member objects should be retained in memory through the lifetime of intended identification in order to minimize setup time.

Operations

Method	Desc	Parameters	Exceptions
UserCollection() void	Creates a UserCollection with a statically-set template capacity (in templates).		
		uint [in] Capacity	<p>DPFP.Error.ErrorCodes.Internal Failed to create User collection.</p> <p>DPFP.Error.ErrorCodes.InvalidParameter Invalid supplied template capacity.</p>
AddUser() void	Adds a User to the User Collection.		
		User [in] User	<p>DPFP.Error.ErrorCodes.InvalidParameter Invalid or corrupt User Collection.</p> <p>DPFP.Error.ErrorCodes.IndexOccupied User already belongs to collection.</p> <p>DPFP.Error.ErrorCodes.NoMemory Cannot add user to collection, Collection template capacity too small.</p> <p>DPFP.Error.ErrorCodes.Internal Template addition failure</p> <p>DPFP.Error.ErrorCodes.InvalidContext template addition failure - out of context operation.</p>

Method	Desc	Parameters	Exceptions
RemoveUser() void	Removes a User from the User Collection		
		UserID [in] ID	<p>DPFP.Error.ErrorCodes.InvalidParameter Invalid supplied UserID.</p> <p>DPFP.Error.ErrorCodes.NotFound Cannot find supplied UserID in collection.</p> <p>DPFP.Error.ErrorCodes.Internal Failed to remove template.</p> <p>DPFP.Error.ErrorCodes.InvalidContext Failed to remove template - out of context operation.</p>
UserExists() bool	Checks whether a User identified by a given ID exists within the User Collection		
		UserID [in] ID	<p>DPFP.Error.ErrorCodes.InvalidParameter Invalid supplied UserID.</p>
User[] User	Returns a User by ID		
		UserID [in] ID	<p>DPFP.Error.ErrorCodes.InvalidParameter Invalid supplied UserID.</p> <p>DPFP.Error.ErrorCodes.NotFound Supplied UserID cannot be found in collection.</p>
ClearUsers() void	Removes all Users from the User Collection		
			<p>DPFP.Error.ErrorCodes.Internal Failed to remove template.</p> <p>DPFP.Error.ErrorCodes.InvalidContext Failed to remove template - out of context operation.</p>
Count() uint	Returns the number of users in collection		

Method	Desc	Parameters	Exceptions
TemplateCapacity() uint	Returns the User Collection's maximum capacity (in templates)		
			DPFP.Error.ErrorCodes.InvalidParameter Invalid template capacity
TemplateCount() uint	Returns the number of templates collected so far		
			DPFP.Error.ErrorCodes.InvalidParameter Invalid template capacity

DPFP.ID.UserID

Type: Class

Description: A unique 256-bit identification associated with a User.

Operations

Method	Desc	Parameters	Exceptions
UserID() void	Constructs a new UserID.		
		byte[] [in] Stream ID	DPFP.Error.ErrorCodes.InvalidParameter Supplied byte array is invalid or incorrectly sized. DPFP.Error.ErrorCodes.Internal Supplied ID is corrupt and cannot be set properly.
UserID() void	Constructs a new UserID from a specified string. Strings longer than 256 bit are truncated		
		string [in] ID	DPFP.Error.ErrorCodes.InvalidParameter Supplied byte array is invalid. DPFP.Error.ErrorCodes.Internal Supplied ID is corrupt and cannot be set properly.
UserID() void	Constructs a new UserID (256 bit in length) from a specified stream		
		Stream [in] Stream	DPFP.Error.ErrorCodes.InvalidParameter Supplied stream is invalid, empty, or corrupt. DPFP.Error.ErrorCodes.Internal Fetched ID is corrupt and cannot be set properly.
UserID() void	Copy-constructs a new UserID		
		UserID [in] Other	DPFP.Error.ErrorCodes.InvalidParameter Supplied byte array is invalid or incorrectly sized. DPFP.Error.ErrorCodes.Internal Supplied ID is corrupt and cannot be set properly.
Bytes() byte[]	Returns unique ID		
ToString() string	Formats UserID to a string		

DPFP.ID.EventHandler

Type: Interface

Description: Defines identification operation events.

Objects implementing the DPFP.ID.EventHandler interface should correctly invoke GUI modifying functions originating in OnStatus and OnIdentify. This can be done through the Invoke() and BeginInvoke() methods of the GUI component's parent form. Please check the Sample application for further information.

Operations

Method	Description	Parameters
OnStatus() void	Fires periodically (roughly 100 msec apart) with the identification progress report	int [in] CandidatesFound Candidates found so far boolean [inout] AbortIdentification Set to true to abort ongoing identification operation.
OnIdentify() void	Fires after a candidate has been found and added to the candidate list	User [in] Candidate Identified candidate boolean [inout] AbortIdentification Set to true to abort ongoing identification operation.

You may redistribute the files in the Redist folder of the One Touch ID SDK product package to your end users pursuant to the terms of the end user license agreement (EULA) located in the Docs folder of the product package.

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 Redist folder of the One Touch ID SDK product package to your end users and to incorporate these files into derivative works for sale and distribution:

- DPOTIDDotNet.msm—One Touch ID redistributable file that contains DPFPSHrNET.dll and DPFPO2MNET.dll
- DPOTIDIdent.msm—One Touch ID redistributable file that contains dpident.dll

Determining Additional Memory Requirements

The following guidelines are for estimating the amount of memory required for identification sets and for the fingerprint templates added to them.

Current implementation requires approximately 40 KB of RAM for each fingerprint template and an additional 12 MB of RAM for each identification set. For instance, performing duplicate enrollment check on two identification sets of 20,000 templates each requires approximately 1624 MB of RAM.

These values are estimates and will vary depending on the actual application in which they are used. For example, the size of an identification set containing 20,000 fingerprint templates may not fit on a system with only 2 GB of installed RAM.

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.

Once converted, template data can be deserialized by One Touch ID's DPFP.Template objects.

Platinum SDK Enrollment Template Conversion for Microsoft Visual C++

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

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

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

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

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

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

    IFPTemplatePtr pIFPTemplate(__uuidof(FPTemplate));

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

Appendix:

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

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

if (error != Er_OK)
return false;

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

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

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

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

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

memcpy(goldBlob, dataGold, blobSizeRequired);

SafeArrayUnaccessData(varValGold.parray);

return true;
}
```

Appendix :

Platinum SDK Enrollment Template Conversion for VB 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 Applications

```
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

candidate

An enrollee whose fingerprint template(s) is determined to be similar to the fingerprint feature set(s).

candidate list

A set of zero, one, or more candidates.

capacity of an identification set

The maximum number of fingerprint templates that an identification set can contain.

comparison

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

context

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

Enrolled User

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

enrollee

See **Enrolled User**.

false-positive identification error rate (FPIR)

The proportion of fingerprint identification transactions by Users not enrolled in the system where a non-empty candidate list is returned.

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 recognition or fingerprint enrollment.

fingerprint data

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

fingerprint data storage subsystem

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

fingerprint enrollment

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

fingerprint feature extraction

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

fingerprint feature set

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

fingerprint identification

The system function that performs a one-to-many comparison, makes decisions, and produces a candidate list.

fingerprint recognition

Either fingerprint verification or fingerprint identification, but does not include duplicate enrollment check.

fingerprint sample

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

fingerprint template

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

fingerprint verification

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

identification operation

A sequence of function calls to perform a fingerprint identification or duplicate enrollment task.

identification set

A non-persistent collection of enrollment information that is used for identification operations.

match

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

non-match

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

one-to-many comparison

The process in which fingerprint feature set(s) from the finger(s) of one User are compared with fingerprint template(s) from the finger(s) of more than one User.

User

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 an enrollee.

Index

A

- additional memory requirements, estimating 36
- additional resources 3
 - online resources 3
 - related documentation 3
- audience for this guide 1

B

- bold typeface, uses of 2

C

- candidate
 - defined 40
- candidate list, defined 40
- chapters, overview of 1
- comparison, defined 40
- compatible fingerprint templates
 - See fingerprint template compatibility matrix
- concepts
 - identification 18
- context, defined 40
- conventions, document
 - See document conventions
- converting Platinum SDK enrollment templates
 - for Microsoft VB 6.0 39
- converting Platinum SDK registration templates to Gold SDK format
 - for Microsoft Visual C++ applications 37
- Courier bold typeface, use of 2

D

- DigitalPersona Developer Connection Forum, URL to 3
- disk drive requirement 4
- document conventions 2
 - notational 2
 - typographical 2
- documentation, related 3
- DPPFP.Data 21
- DPPFP.Data.FeatureSet 22
- DPPFP.Data.Template 22
- DPPFP.Error.ErrorCodes 23
- DPPFP.Error.ExceptionFactory 24
- DPPFP.Error.SDKException 24
- DPPFP.ID.Candidate 25
- DPPFP.ID.CandidateIDList 25
- DPPFP.ID.CandidateList 25
- DPPFP.ID.EventHandler 35
- DPPFP.ID.FingerPosition 26

- DPPFP.ID.Identification 27
- DPPFP.ID.User 29
- DPPFP.ID.UserCollection 31
- DPPFP.ID.UserID 34

E

- Enrolled User 40
- enrollee, defined 18, 40
 - See fingerprint enrollment subject

F

- false-positive identification error rate, defined 40
- finger positions, mapping
 - URLs to obtain standards for 3
- fingerprint characteristic, defined 40
- fingerprint data storage subsystem, defined 40
- fingerprint data, defined 40
- fingerprint enrollment, defined 40
- fingerprint feature extraction, defined 40
- fingerprint feature set, defined 18, 40
- fingerprint identification
 - defined 40
- fingerprint recognition
 - defined 40
 - guide to 3
- fingerprint sample, defined 40
- fingerprint template
 - defined 18, 41
 - estimating memory requirements for 36
 - fingerprint template compatibility matrix 5
 - fingerprint verification, defined 41
- FPIR
 - See false-positive identification error rate
- functions
 - Identification API
 - definitions of 21
 - See also individual functions by name

G

- getting started 6
- Gold SDK developer guide 3

H

- hard disk
 - available space requirements 4
 - SDK files and folders installed on 10

I

- identification

- defined 18
- Identification API
 - functions
 - definitions of 21
 - See also individual functions by name
 - reference 21
 - See `dpIdent.h`
- identification operation
 - defined 18, 41
- Identification Sample application 6
- identification set
 - defined 41
 - estimating memory requirements for 36
- identification terminology and concepts 18
- identification, overview of 14
- important notation, defined 2
- installing
 - the runtime environment 11
 - the SDK 10
- introduction to this guide 1
- italics typeface, uses of 2
- L**
- licensing requirements
 - redistributable files 36
- M**
- mapping finger positions
 - URLs to obtain standards for 3
- match, defined 41
- memory requirements
 - estimating additional 36
 - minimum 4
- merge modules
 - licensing requirements for redistribution of 36
- Microsoft Visual Studio, system requirement 4
- N**
- non-match, defined 41
- notational conventions 2
- note notation, defined 2
- O**
- One Touch ID identification
 - See identification
- One Touch ID Runtime Environment
 - See runtime environment
- One Touch ID SDK Developer Guide, introduction to 1
- One Touch ID SDK, installing 10
- one-to-many comparison, defined 41
- online resources 3
- overview

- of chapters 1
- of identification 14
- P**
- Platinum SDK developer guide 3
- Platinum-to-Gold SDK template conversion 37
- processor requirements 4
- product compatibility
 - See fingerprint template compatibility matrix

- Q**
- quick start guide 6

- R**
- RAM requirements
 - See memory requirements
- redistributable files
 - licensing requirements for redistribution of 36
- redistributing One Touch ID SDK components 36
- requirements, system
 - See system requirements
- resources, additional
 - See additional resources
- resources, online
 - See online resources
- RTE
 - See runtime environment
- runtime environment
 - installing 11
 - See also Identification DLL and Licensing DLL

- S**
- sample code for converting Platinum SDK enrollment templates
 - Microsoft VB 6.0 39
- sample code for converting Platinum SDK registration templates to Gold SDK format
 - for Microsoft Visual C++ applications 37
- system requirements 4
 - available hard-disk space 4
 - disk drive 4
 - memory 4
 - Microsoft Visual Studio 4
 - processors 4

- T**
- target audience for this guide 1
- template compatibility matrix
 - See fingerprint template compatibility matrix
- terminology
 - identification 18
- typefaces, uses of

Index

- bold 2

- Courier bold 2

- italics 2

- typographical conventions 2

U

- updates for DigitalPersona software products, URL for downloading 3

URLs

- DigitalPersona Developer Connection Forum 3

- standards for mapping fingers to finger number positions 3

- Updates for DigitalPersona Software Products 3

- User 41

- Users and User Collections 18

V

- Visual Studio

- See Microsoft Visual Studio

W

- warning notation, defined 2