

DigitalPersona, Inc.

DigitalPersona Pro[®] SDK

Version 4.2

Developer Guide



digitalPersona.

DigitalPersona, Inc.

© 1996–2007 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, U.are.U, DigitalPersona Pro, and One Touch 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, Active Directory, Internet Explorer, Visual C++, Windows, Windows NT, and Windows Server are registered trademarks and Windows Vista is a trademark of Microsoft Corporation in the United States and other countries.

This DigitalPersona Pro 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

The DigitalPersona Web site provides an online technical support form at <http://www.digitalpersona.com/support/enterprise/chooseproduct.php>. Simply describe your issue and include your contact information, and a technical support representative will contact you shortly by email or by phone.

Phone support is available at (877) 378-2740 in the U.S. only. Outside the U.S., call +1 650-474-4000.

Feedback

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

TechPubs@digitalpersona.com

or

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

Table of Contents

| | | |
|---|---------------------------------------------------|----|
| 1 | Introduction | 1 |
| | Target Audience | 1 |
| | Chapter Overview | 1 |
| | Document Conventions | 2 |
| | Notational Conventions | 2 |
| | Typographical Conventions | 2 |
| | Naming Conventions | 2 |
| | Additional Resources | 3 |
| | Related Documentation | 3 |
| | Online Resources | 3 |
| | System Requirements | 3 |
| | Hardware | 3 |
| | Software | 4 |
| | Supported DigitalPersona Products | 4 |
| 2 | Installation | 5 |
| | Installing the SDK | 5 |
| | Contents of the Pro SDK Product CD | 6 |
| 3 | Overview | 7 |
| | Pro SDK Terminology and Concepts | 7 |
| | Old and New Terms | 7 |
| | Operations | 7 |
| | Priorities | 8 |
| | User Interface Support | 9 |
| | Pro SDK Data Flow | 10 |
| | Operation Workflows | 11 |
| | Fingerprint Enrollment Operation Workflows | 11 |
| | Fingerprint Acquisition Operation Workflows | 15 |
| 4 | API Reference | 24 |
| | Functions | 24 |
| | DPFPAddUserToIdentList | 28 |
| | DPFPAuthenticate | 28 |
| | DPFPBufferFree | 30 |
| | DPFPCommitRegistrationChanges | 30 |
| | DPFPCreateAcquisition | 31 |
| | DPFPCreateRegistration | 35 |

| | |
|-----------------------------------|----|
| DPFPDeleteFinger | 38 |
| DPFPDeleteSecret | 39 |
| DPFPDestroyAcquisition | 40 |
| DPFPDestroyRegistration | 40 |
| DPFPEnumerateDevices | 41 |
| DPFPGetDeviceInfo | 42 |
| DPFPGetUserInfo | 42 |
| DPFPGetVersion | 44 |
| DPFPIdentify | 44 |
| DPFPIdentifyUser | 45 |
| DPFPInit | 46 |
| DPFPisRegistrationAllowed | 47 |
| DPFPisRegistrationAllowedEx | 48 |
| DPFPStartAcquisition | 49 |
| DPFPStartRegistration | 49 |
| DPFPStopAcquisition | 51 |
| DPFPStopRegistration | 51 |
| DPFPTerm | 52 |
| DPFPWriteSecretContent | 52 |
| DPSDBufferFree | 53 |
| DPDSDIdentifyLogonUser | 54 |
| DPDSDIdentifyUser | 55 |
| DPDSDVerifyLogonUser | 57 |
| DPDSDVerifyUser | 58 |
| Data Structures | 61 |
| DP_AUTHENTICATION_INFO | 61 |
| DP_DEVICE_INFO | 62 |
| DP_DEVICE_VERSION | 63 |
| DP_HW_INFO | 63 |
| DP_PRODUCT_VERSION | 64 |
| DP_USER_INFO_0 | 64 |
| Data Types and Constants | 65 |
| HDPOPERATION | 65 |
| HDPCREDENTIAL | 65 |
| DP_MAX_USB_STRING_SIZE | 65 |
| | |
| A Comparative Glossary | 66 |
| | |
| Glossary | 67 |
| | |
| Index | 70 |

The DigitalPersona Pro® SDK provides developers with simple, powerful tools to extend DigitalPersona Pro 4.2 with custom applications. Developers can fingerprint-enable access to their applications by leveraging DigitalPersona Pro security, credential management in Microsoft® Active Directory®, user interface, and deployment tools.

The DigitalPersona Pro SDK is designed to work with DigitalPersona Pro Server and DigitalPersona Pro Workstation, which run with the Microsoft® Windows® operating system and within a Microsoft-enabled distributed server environment. The standard shipping version of the SDK requires a DigitalPersona U.are.U® fingerprint reader for fingerprint scanning. Custom SDKs using the core DigitalPersona fingerprint recognition technology are also available for other fingerprint readers.

The DigitalPersona Pro SDK contains documentation and easy-to-follow sample code that enable you to quickly build DigitalPersona Pro functionality into your application.

Target Audience

This guide is for developers who have a working knowledge of the C and C++ programming languages. In addition, familiarity with DigitalPersona Pro 4.2 and Microsoft Active Directory is recommended.

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 Pro SDK; identifies the minimum system requirements needed to run the Pro SDK; and lists the DigitalPersona products supported by the Pro SDK.

Chapter 2, Installation, contains instructions for installing various components of the product, identifies the files and folders that are installed on your hard disk, and describes the contents of the Pro SDK product CD.

Chapter 3, Overview, introduces Pro SDK terminology and concepts; shows how data flows among the DigitalPersona Pro Workstation and Server, the fingerprint-enabled application, and Microsoft Active Directory; and includes typical workflow diagrams and explanations of the Pro API functions used to perform the tasks in the workflows.

Chapter 4, API Reference, defines the functions, data structures, and some of the data types and constants that are part of the Pro API.

Appendix A, Comparative Glossary, compares some of the terms used in the *DigitalPersona Pro for Active Directory Administrator Guide* with corresponding new terms used in this guide.

A main 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.

WARNING: Warnings alert you to actions that can cause loss of data or system crashes.

Typographical Conventions

The following typographical conventions are used in this guide:

| Typeface | Purpose | Example |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Courier bold | Used to indicate computer programming code | The wParam of the message specifies the event type. Initialize and allocate necessary resources by calling the DPFPInit function. |
| <i>Italics</i> | Used for emphasis or to introduce new terms If you are viewing this document online, clicking on text in italics may also activate a hypertext link to other areas in this guide. | You <i>must</i> call the DPFPBufferFree function to free allocated memory. (emphasis) <i>A fingerprint</i> is an impression of the ridges on the skin of a finger. (new term) See <i>Initialization</i> on page 40. (link to heading and page) |

Naming Conventions

The *DPFP* prefix used in Pro API functions stands for *DigitalPersona Fingerprint*.

The *DPSD* prefix used in Pro API functions stands for *DigitalPersona Super Duper*.

The *DP* prefix used in Pro API data structures, data types, and constants stands for *DigitalPersona*.

Additional Resources

You can refer to the resources in this section to assist you in using the Pro SDK.

Related Documentation

| Subject | Document |
|--------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| <i>DigitalPersona Pro for Active Directory Administrator Guide</i> | A comprehensive resource for information about DigitalPersona Pro for Active Directory |
| Late-breaking news about the product | Readme.txt files provided in the root directory of the product CD as well as in some subdirectories |

Online Resources

| Web Site Name | URL |
|-------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| DigitalPersona Developer Connection Forum for DigitalPersona Developers | http://www.digitalpersona.com/webforums/ |
| Latest updates for DigitalPersona software products | http://www.digitalpersona.com/support/downloads/software.php |

System Requirements

This section lists the minimum hardware and software requirements needed to run the Pro SDK.

Hardware

- X86-based processor or better
- 16 MB of RAM
- 5 MB of available hard-disk space
- CD-ROM drive
- USB port on the computer that the fingerprint reader is connected to
- DigitalPersona U.are.U 4000B Fingerprint Reader, Revision 100 or 101

Software

- Microsoft® Windows® 2000 Standard or Enterprise; Microsoft® Windows® XP Professional or Embedded (Home edition is not supported); Microsoft® Windows Server® 2003 or Microsoft Windows Small Business Server 2003 R2; or Microsoft® Windows Vista™ Business, Ultimate, or Enterprise, 32- or 64-bit versions (Home and Home Premium editions are not supported)
- Microsoft Active Directory
- DigitalPersona Pro Workstation 4.2
- Microsoft® Internet Explorer® 6 or later (required for One Touch® SignOn and One Touch Internet)

Supported DigitalPersona Products

The Pro SDK supports the following DigitalPersona products:

- DigitalPersona Pro for Active Directory 4.2
- DigitalPersona U.are.U 4000B Fingerprint Reader, Revisions 100 and 101
- DigitalPersona U.are.U Fingerprint Keyboard

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

Installing the SDK

To install the Pro SDK

1. Insert the Pro SDK product CD into your CD-ROM drive.
2. Double-click Setup.exe.
3. Follow the installation instructions as they appear.

Table 1 describes the files and folders that are installed in the <destination folder>\Pro SDK folder on your hard disk.

Table 1. Pro SDK installed files and folders

| Folder | File | Description |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------|
| Docs | Pro SDK Developer Guide.pdf | <i>DigitalPersona Pro SDK Developer Guide</i> |
| Include | DPFPApi.h | Pro API definition |
| | DPSDApi.h | Pro API with UI support definitions |
| Lib | DPFPApi.lib | Import library for DPFPApi.dll |
| | DPSDApi.lib | Import library for DPSDApi.dll |
| Samples\C++ Samples\Authenticate | This folder contains a sample Microsoft® Visual C++® project that shows you how to use the Pro API. | |
| Samples\C++ Samples\Authenticate\Release | Authenticate.exe | Sample fingerprint-enabled application |
| Samples\C++ Samples\AuthenticateWithUISupport | This folder contains a sample Microsoft Visual C++ project that shows you how to use the Pro API with UI support. | |
| Samples\C++ Samples\AuthenticateWithUISupport\Release | AuthenticateWithUISupport.exe | Sample fingerprint-enabled application with UI support |

Contents of the Pro SDK Product CD

Table 2 describes the files and folders on the Pro SDK product CD.

Table 2. Pro SDK product CD files and folders

| Folder | File | Description |
|-----------------------------------|--------------------------------------|---------------------------------------------------------------------------------------|
| | AUTORUN.INF | Automatically launches the SDK installer |
| | Readme.txt | Information that supplements the DigitalPersona Pro SDK documentation |
| | Setup.exe | Installs the Pro SDK |
| Docs | Pro SDK Developer Guide.pdf | <i>DigitalPersona Pro SDK Developer Guide</i> |
| | SDK EULA_Elec_070783.doc | End User License Agreement for DigitalPersona Software Development Kit (SDK) Products |
| Install | Setup.exe | Installs the Pro SDK |
| | Setup.msi | Pro SDK installation package |
| Third-Party Software\Adobe Reader | AdbeRdr80_en_US.exe | Installs Adobe® Reader® 8 |
| | Readme.txt | Adobe Reader notice |
| Third-Party Software\MSI 3.1 | WindowsInstaller-KB893803-v2-x86.exe | Microsoft Windows installer package |

This chapter introduces Pro SDK terminology and concepts; shows how data flows among the DigitalPersona Pro Workstation and Server, the fingerprint-enabled application, and Microsoft Active Directory; and includes typical workflow diagrams and explanations of the Pro API functions used to perform the tasks in the workflows.

Pro SDK Terminology and Concepts

This section defines the Pro SDK terminology and concepts that are used throughout this guide.

Old and New Terms

This section is for those who are already familiar with the terminology and concepts presented in the *DigitalPersona Pro for Active Directory Administrator Guide*. Some of the terms used in this guide, although similar in meaning to the terms used in the administrator guide, have been changed. Appendix A on page 66 contains a comparative glossary for your reference. In addition, old terms are cross-referenced to new terms in the main glossary that begins on page 67.

Operations

Each time a user touches a fingerprint reader with a finger and the fingerprint reader successfully scans the fingerprint, the reader sends a fingerprint image to the Local Biometric Authentication Service (BAS). The Local BAS then dispatches this message to the fingerprint-enabled application (FEA). Only one FEA can obtain the results of a single fingerprint scan. (The *fingerprint-enabled application* is the application that you develop using Pro SDK API functions.)

When the FEA is required to perform an action in response to a successful fingerprint scan, it should begin by creating one of the following *operations*:

1. Fingerprint enrollment operation

A fingerprint enrollment operation is created for the purpose of enrolling a user's fingerprint. When the successful-operation message is received by the FEA, a *stored fingerprint credential* is created, added to the user record, and stored for later comparison with a *supplied fingerprint credential*. This operation can also be used to delete a stored fingerprint credential from a user record.

2. Fingerprint acquisition operation

A *fingerprint acquisition operation* is created for the purpose of acquiring a *supplied fingerprint credential* to be used for

- Performing *fingerprint authentication*, which is the process of comparing a supplied fingerprint credential with a stored fingerprint credential based on the user name and a fingerprint provided by the user. If the two fingerprint credentials match, the operation returns the DigitalPersona Pro Secret or a message confirming that the fingerprint is from the user.
- Performing *fingerprint identification*, which is the process of comparing a supplied fingerprint credential with one or more enrollment fingerprint credentials based on a fingerprint provided by the user. If a matching stored fingerprint credential is found, the operation returns the user name.
- Managing user data, which includes retrieving information about a user, checking if a user can be enrolled in the system, and adding or deleting stored fingerprint credentials.
- Managing secrets, which includes writing or deleting secret content from a user record.

Priorities

A fingerprint enrollment operation always has normal priority level. However, during the creation of a fingerprint acquisition operation, the FEA specifies a priority level, which can be low, normal, or high. It is possible to create and subscribe to any number of fingerprint acquisition operations with normal priority, but only one operation for each of low and high priority can be created and subscribed to at a time.

When the Local BAS is ready to dispatch the results of the fingerprint scan, it processes operations using the following hierarchy:

- If there is a high-priority operation subscribed to, the result is dispatched to the process that owns the operation.
- If there is no high-priority operation subscribed to, the Local BAS determines which process owns the topmost window. If there is a normal-priority operation owned by this process, it receives the result.
- If the first two steps do not allow the Local BAS to dispatch the result, the process owning the low-priority operation (if subscribed to) receives the result.
- If the result is still not dispatched, it is discarded.

User Interface Support

The Pro SDK provides C wrappers with user interface (UI) support. These wrappers encompass the functions that return the user data required for fingerprint authentication and fingerprint identification. This feature also provides customizable DigitalPersona Pro dialog boxes that invite users to verify their identities.

Fingerprint Authentication

Fingerprint authentication is started by calling the **DPSDVerifyUser** function (*page 58*). A dialog box is displayed inviting the user to touch the fingerprint reader. In addition, the user must provide a user name. You provide this functionality in your application. (You can change dialog box's title and default text.) If the function succeeds, the dialog box closes, and the handle to a supplied fingerprint credential is returned. This handle can be used later, for example, in a call to the **DPFPAuthenticate** function (*page 28*).

Fingerprint Identification

Fingerprint identification is started by calling the **DPSDIdentifyUser** function (*page 55*). A dialog box is displayed inviting the user to touch the fingerprint reader. (You can change the dialog box's title and default text.) If the function succeeds, the dialog box closes, and the user name and the handle to a supplied fingerprint credential are returned. This handle can be used later, for example, in a call to the **DPFPAuthenticate** function (*page 28*).

If the user is not identified, the user is notified via the dialog box to enter a user name and touch the fingerprint reader again. When this happens, fingerprint authentication is used instead of fingerprint identification.

Fingerprint Authentication and Identification for the Logon User

The **DPSDIdentifyLogonUser** (*page 54*) and **DPSDVerifyLogonUser** (*page 57*) functions are similar to the **DPSDIdentifyUser** and **DPSDVerifyUser** functions, respectively. The difference between them is that the **LogonUser** functions return a handle to a token that represents the logged-on user, which can then be used to impersonate the user or to create a process that runs in the context of the user. If the user's Windows logon password does not exist in the DigitalPersona database, the user is asked to provide the password, which is verified and stored in the database.

The **DPSDIdentifyLogonUser** function displays a DigitalPersona Pro user interface for fingerprint identification with authentication. This function requires that the user provide a fingerprint. However, instead of returning the user name, the function retrieves the DigitalPersona Pro Secret, which is the user's logon password, from the DigitalPersona database. If the **DPSDIdentifyLogonUser** function succeeds, it returns a handle to a token that represents the logged-on user, performs user logon, and closes the dialog box.

The **DPSDVerifyLogonUser** function displays a DigitalPersona Pro user interface for fingerprint authentication. This function requires that the user provide a user name and a fingerprint. The function retrieves the DigitalPersona Pro Secret, which is the user's logon password, from the DigitalPersona database. If the **DPSDVerifyLogonUser** function succeeds, it returns a handle to a token that represents the logged-on user, performs user logon, and closes the dialog box.

Pro SDK Data Flow

Figure 1 illustrates how data flows among the DigitalPersona Pro Workstation and Server, the FEA, and Active Directory. The figure is followed by the steps in the typical Pro SDK authentication operation, which is remote fingerprint authentication. During fingerprint identification or when fingerprint authentication takes place on a local machine, data is passed within the DigitalPersona Pro Workstation framework only.

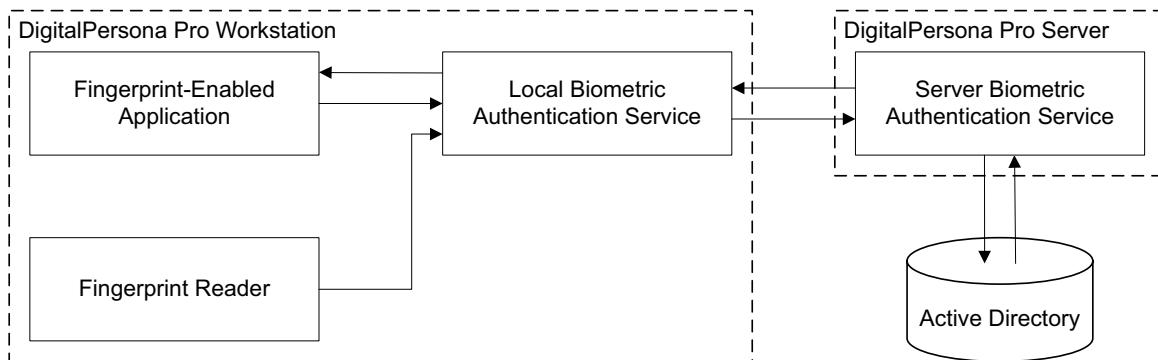


Figure 1. Typical Pro SDK fingerprint authentication operation

IMPORTANT: To perform fingerprint authentication without first performing fingerprint identification, a user name is required. You must build the functionality for acquiring the user name into your application, as it is not part of the Pro API.

1. The FEA initiates an authentication operation and invites the user to provide a fingerprint by touching the fingerprint reader.
2. The user touches the fingerprint reader, and the fingerprint reader sends a fingerprint image to the Local BAS.
3. The Local BAS receives the fingerprint image, creates a supplied fingerprint credential from the fingerprint image, and sends the handle to the supplied fingerprint credential to the FEA.
4. The FEA receives the handle and sends a request to the Local BAS for the DigitalPersona Pro Secret. This request includes the user name and the handle to the supplied fingerprint credential.
5. The Local BAS receives the request from the FEA, generates a new request for the DigitalPersona Pro Secret, and sends it to the Server BAS. This request includes the user name, the supplied fingerprint credential, and the name of the requested DigitalPersona Pro Secret.
6. The Server BAS receives the request from the Local BAS and sends a request for a stored fingerprint credential to Active Directory.
7. Active Directory returns the stored fingerprint credential.

8. The Server BAS compares the supplied fingerprint credential from the Local BAS with the stored fingerprint credential from Active Directory. If the two match, the Server BAS sends a request for the DigitalPersona Pro Secret to Active Directory.
9. Active Directory returns the requested DigitalPersona Pro Secret to the Server BAS.
10. The Server BAS releases the DigitalPersona Pro Secret to the Local BAS.
11. The Local BAS returns the DigitalPersona Pro Secret to the FEA.

Operation Workflows

Typical operation workflows are presented in this section. Three subsections contain illustrations of the workflows, which are accompanied by explanations of the Pro API functions used to perform the tasks in each of the following workflows:

- Fingerprint enrollment operation

This subsection contains two workflows: one for creating a stored fingerprint credential and one for deleting a stored fingerprint credential.

- Fingerprint acquisition operation for fingerprint authentication

This subsection contains two workflows: one without UI support and one with UI support.

- Fingerprint acquisition operation for fingerprint identification with fingerprint authentication

This subsection contains two workflows: one without UI support and one with UI support.

Fingerprint Enrollment Operation Workflows

This section contains two fingerprint enrollment workflows: one for creating a stored fingerprint credential and one for deleting a stored fingerprint credential.

Creating a Stored Fingerprint Credential

The typical fingerprint enrollment operation workflow for creating a stored fingerprint credential is illustrated *Figure 2* and is followed by explanations of the Pro API functions used to perform the tasks in the workflow.

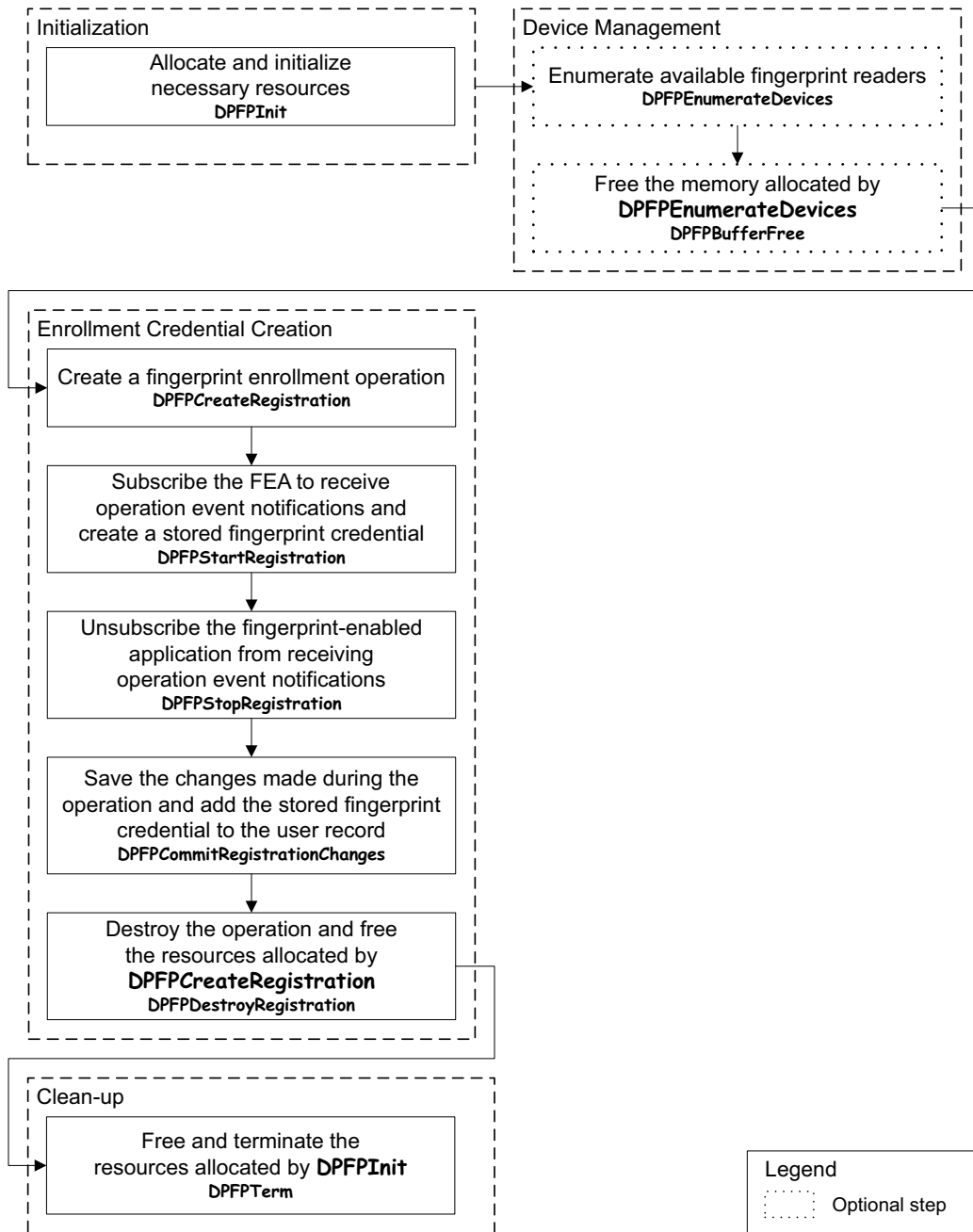


Figure 2. Typical fingerprint enrollment operation workflow: Creating a stored fingerprint credential

Initialization

- Initialize and allocate necessary resources by calling the **DPFPInit** function (page 46).

Device Management

Steps 1 and 2 are optional; however, if you perform step 1, you must also perform step 2.

1. Enumerate the available fingerprint readers (devices) connected to a computer by calling the **DPFPEnumerateDevices** function (page 41).
2. Free the memory allocated by the **DPFPEnumerateDevices** function by calling the **DPFPBufferFree** function (page 30).

Enrollment Credential Creation

1. Create a fingerprint enrollment operation by calling the **DPFPCreateRegistration** function (page 35).
2. Subscribe the FEA to receive fingerprint enrollment operation event notifications by calling the **DPFPStartRegistration** function (page 49).

When the **WN__REGISTRATION_COMPLETED** message is received, a stored fingerprint credential is created.

3. Unsubscribe the FEA from receiving operation event notifications by calling the **DPFPStopRegistration** function (page 51).
4. Save the changes made in steps 2 and 3, add the stored fingerprint credential to the user record, and store it by calling the **DPFPCommitRegistrationChanges** function (page 30).
5. Destroy the fingerprint enrollment operation and free the resources allocated by the **DPFPCreateRegistration** function by calling the **DPFPDestroyRegistration** function (page 40).

Clean-up

- Free and terminate the resources allocated by the **DPFPInit** function by calling the **DPFPTerm** function (page 52).

Deleting a Stored Fingerprint Credential

The typical fingerprint enrollment operation workflow for deleting a stored fingerprint credential is illustrated in *Figure 3* and is followed by explanations of the Pro API functions used to perform the tasks in the workflow.

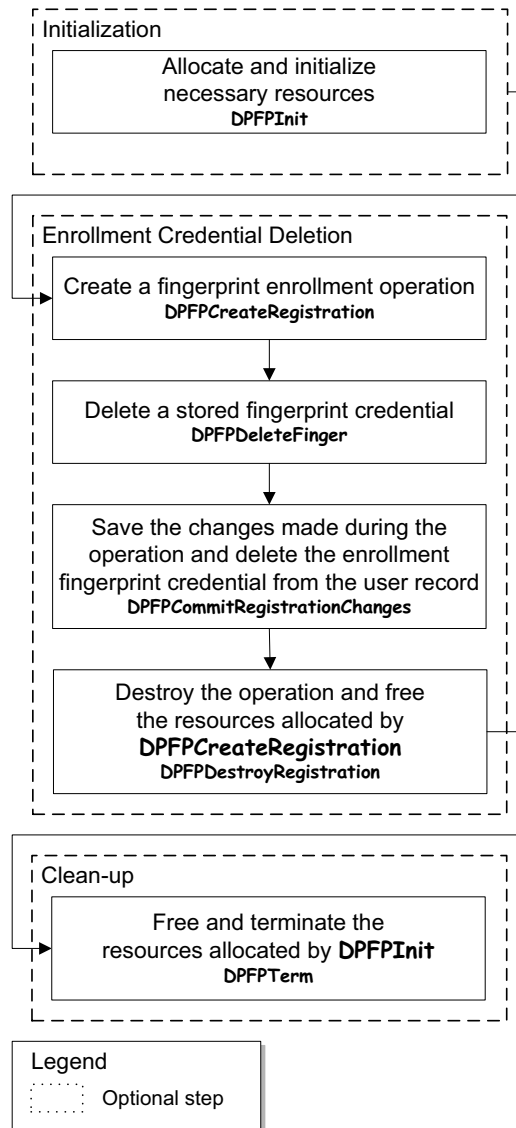


Figure 3. Typical fingerprint enrollment operation workflow: Deleting a stored fingerprint credential

Initialization

- Initialize and allocate necessary resources by calling the **DPFPInit** function (*page 46*).

Enrollment Credential Deletion

1. Create a fingerprint enrollment operation by calling the **DPFPCreateRegistration** function (*page 35*).
2. Delete a stored fingerprint credential by calling the **DPFPDeleteFinger** function (*page 38*).
3. Save the changes made in step 2, and delete the fingerprint credential from the user record by calling the **DPFPCommitRegistrationChanges** function (*page 30*).
4. Destroy the fingerprint enrollment operation and free the resources allocated by the **DPFPCreateRegistration** function by calling the **DPFPDestroyRegistration** function (*page 40*).

Clean-up

- Free and terminate the resources allocated by the **DPFPInit** function by calling the **DPFPTerm** function (*page 52*).

Fingerprint Acquisition Operation Workflows

This section contains workflows for the fingerprint acquisition operation: two for performing fingerprint authentication—with and without UI support—and two for performing fingerprint identification with fingerprint authentication—with and without UI support.

Performing Fingerprint Authentication without UI Support

The typical fingerprint authentication workflow without UI support is illustrated in *Figure 4* and is followed by explanations of the Pro API functions used to perform the tasks in the workflow.

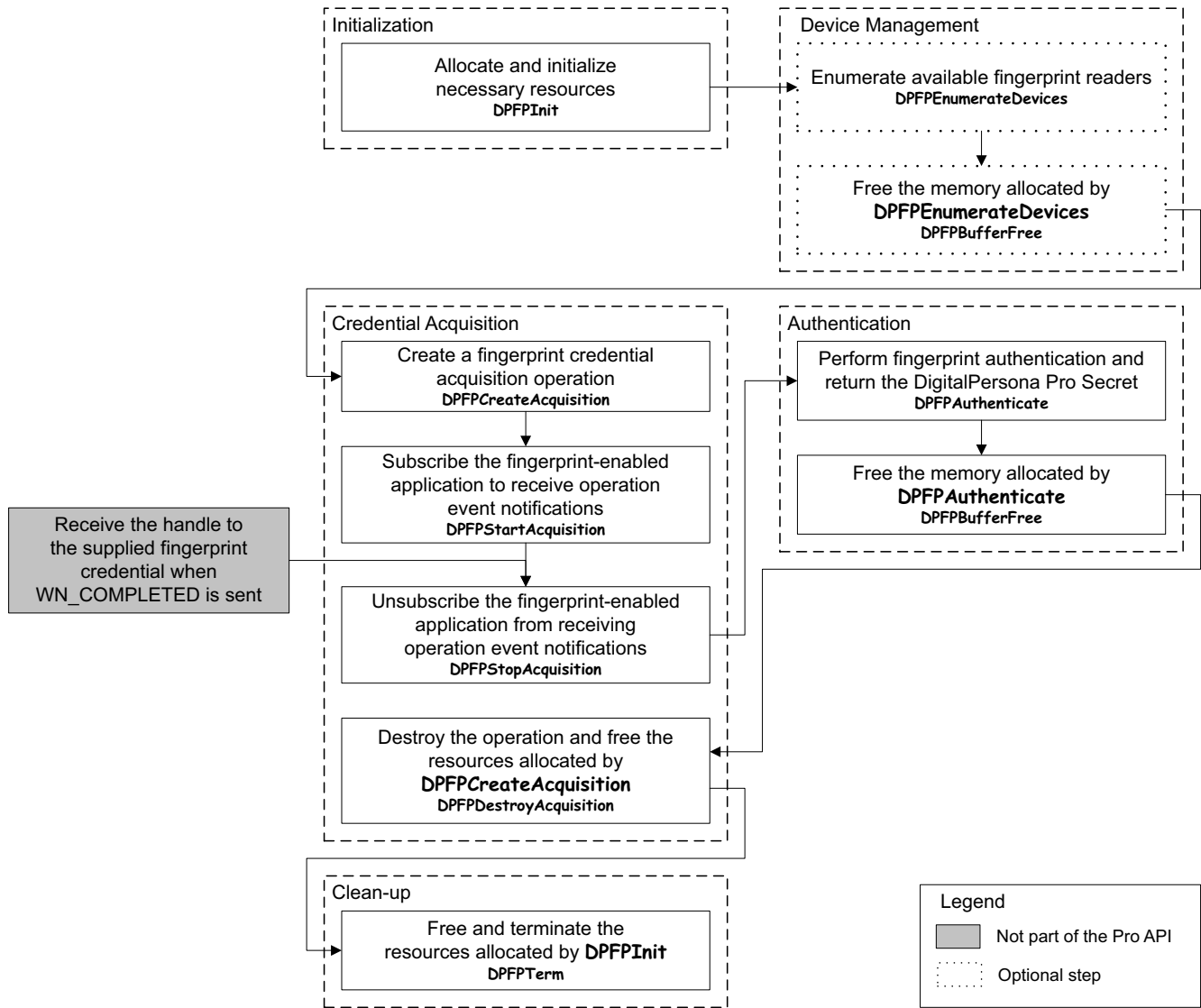


Figure 4. Typical fingerprint authentication workflow: Without UI Support

Initialization

- Initialize and allocate necessary resources by calling the `DPFPInit` function (page 46).

Device Management

Steps 1 and 2 are optional; however, if you perform step 1, you must also perform step 2.

1. Enumerate the available fingerprint readers (devices) connected to a computer by calling the `DPFPEnumerateDevices` function (page 41).
2. Free the memory allocated by the `DPFPEnumerateDevices` function by calling the `DPFPBufferFree` function (page 30).

Credential Acquisition

IMPORTANT: To perform fingerprint authentication without first performing fingerprint identification, a user name is required. You must build the functionality for acquiring the user name into your application, as it is not part of the Pro API.

1. Create a fingerprint acquisition operation by calling the `DPFPCreateAcquisition` function (page 31).
2. Subscribe the FEA to receive fingerprint acquisition operation event notifications by calling the `DPFPStartAcquisition` function (page 49).

3. Acquire a fingerprint credential from the fingerprint reader.

NOTE: This functionality is provided by the fingerprint reader and is not part of the Pro API.

A supplied fingerprint credential is created, and the handle to the fingerprint credential is passed to the `DPFPAuthenticate` function.

4. Unsubscribe the FEA from receiving fingerprint acquisition operation event notifications by calling the `DPFPStopAcquisition` function (page 51).

Authentication

1. Perform fingerprint authentication and return the DigitalPersona Pro Secret by calling the `DPFPAuthenticate` function (page 28).
2. Free the memory allocated by the `DPFPAuthenticate` function by calling the `DPFPBufferFree` function (page 30).
3. Destroy the fingerprint acquisition operation and free the resources allocated by the `DPFPCreateAcquisition` function by calling the `DPFPDestroyAcquisition` function (page 40).

Clean-up

- Free and terminate the resources allocated by the `DPFPInit` function by calling the `DPFPTerm` function (page 52).

Performing Fingerprint Authentication with UI Support

The typical fingerprint authentication workflow with UI support is illustrated in *Figure 5* and is followed by explanations of the Pro API functions used to perform the tasks in the workflow.

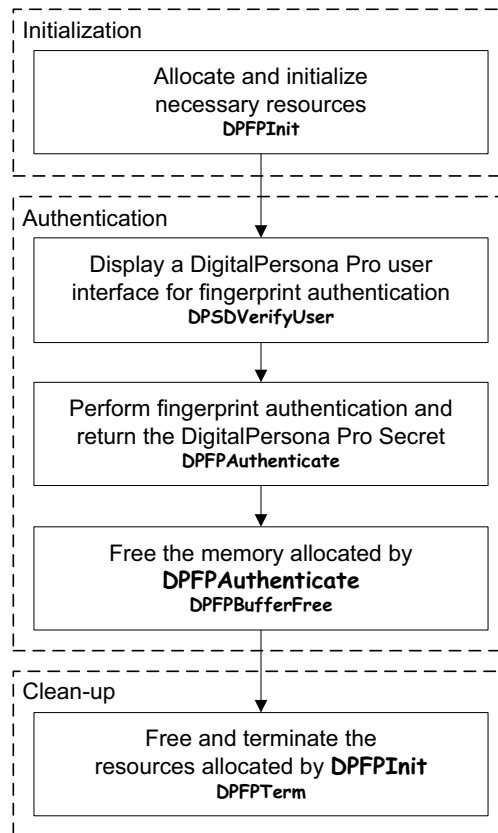


Figure 5. Typical fingerprint authentication workflow: With UI Support

Initialization

- Initialize and allocate necessary resources by calling the **DPFPInit** function (*page 46*).

Authentication

1. Display a DigitalPersona Pro user interface for fingerprint authentication, and return the handle to the supplied fingerprint credential by calling the **DPSDVerifyUser** function (*page 58*).

The handle to the supplied fingerprint credential is passed to the **DPFPAuthenticate** function.

2. Perform fingerprint authentication and return the DigitalPersona Pro Secret by calling the **DPFPAuthenticate** function (*page 28*).
3. Free the memory allocated by calls to the **DPFPAuthenticate** function by calling the **DPFPBufferFree** function (*page 30*).

Clean-up

- Free and terminate the resources allocated by the **DPFPInit** function by calling the **DPFPTerm** function (*page 52*).

Performing Fingerprint Identification with Authentication without UI Support

The typical fingerprint identification with fingerprint authentication workflow without UI support is illustrated in *Figure 6* and is followed by explanations of the Pro API functions used to perform the tasks in the workflow.

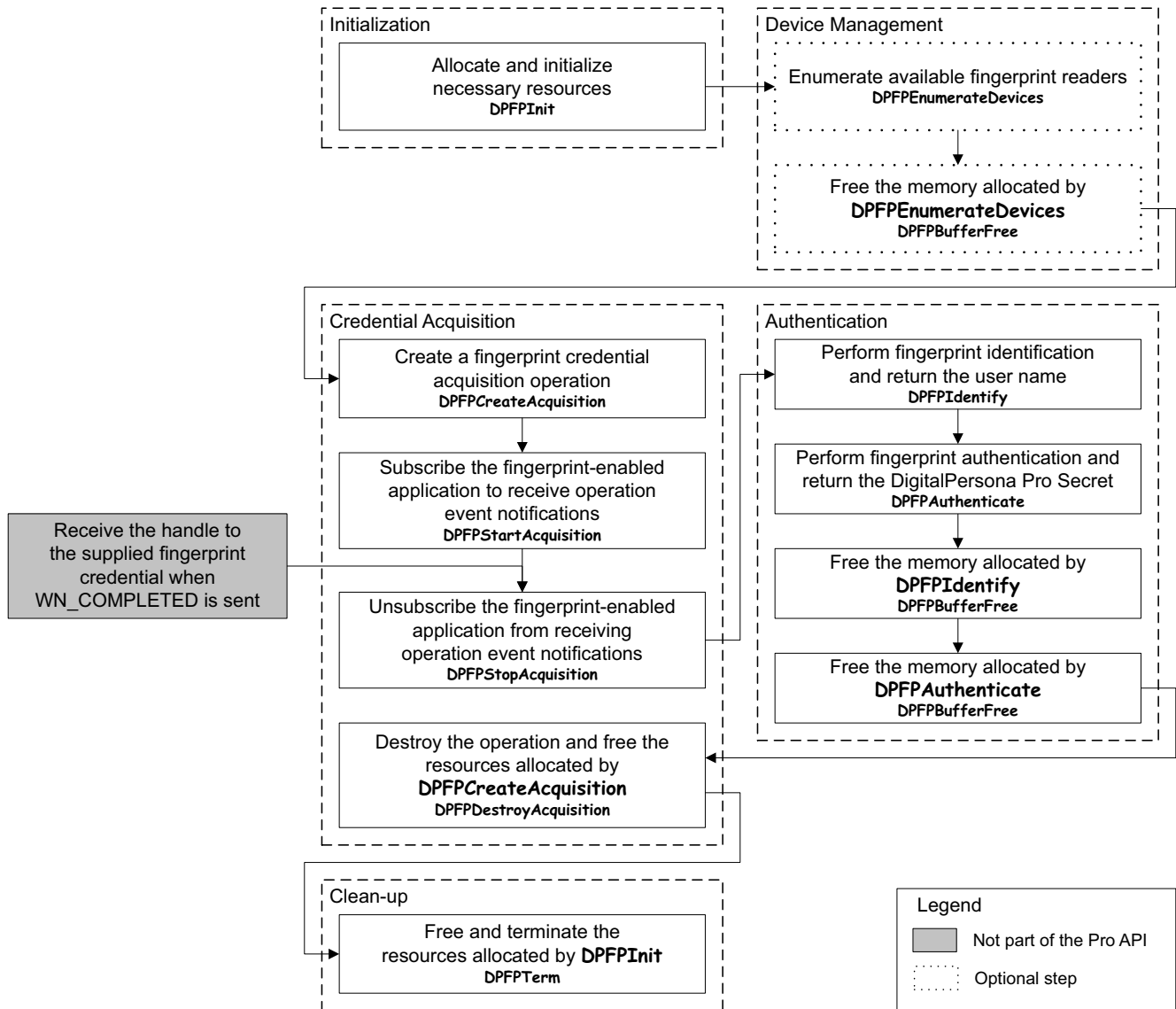


Figure 6. Typical fingerprint identification with fingerprint authentication workflow: Without UI Support

Initialization

- Initialize and allocate necessary resources by calling the **DPFPInit** function (page 46).

Device Management

Steps 1 and 2 are optional; however, if you perform step 1, you must also perform step 2.

1. Enumerate the available fingerprint readers (devices) connected to a computer by calling the **DPFPEnumerateDevices** function (page 41).
2. Free the memory allocated by the **DPFPEnumerateDevices** function by calling the **DPFPBufferFree** function (page 30).

Credential Acquisition

1. Create a fingerprint acquisition operation by calling the **DPFPCreateAcquisition** function (page 31).
2. Subscribe the FEA to receive fingerprint acquisition operation event notifications by calling the **DPFPStartAcquisition** function (page 49).

3. Acquire a fingerprint credential from the fingerprint reader.

NOTE: This functionality is provided by the fingerprint reader and is not part of the Pro API.

A supplied fingerprint credential is created, and the handle to the fingerprint credential is passed to the **DPFPIdentify** function.

4. Unsubscribe the FEA from receiving fingerprint acquisition operation event notifications by calling the **DPFPStopAcquisition** function (page 51).

Identification and Authentication

1. Perform fingerprint identification, and return the user name by calling the **DPFPIdentify** function (page 44).

The user name and the handle to the supplied fingerprint credential are passed to the **DPFPAuthenticate** function.

2. Perform fingerprint authentication and return the DigitalPersona Pro Secret by calling the **DPFPAuthenticate** function (page 28).
3. Free the memory allocated by the **DPFPIdentify** function by calling the **DPFPBufferFree** function (page 30).
4. Free the memory allocated by the **DPFPAuthenticate** function by calling the **DPFPBufferFree** function (page 30).

5. Destroy the fingerprint acquisition operation and free the resources allocated by the **DFFPCreateAcquisition** function by calling the **DFFPDestroyAcquisition** function (page 40).

Clean-up

- Free and terminate the resources allocated by the **DFFPInit** function by calling the **DFFPTerm** function (page 52).

Performing Fingerprint Identification with Authentication with UI Support

The typical fingerprint identification with authentication workflow with UI support is illustrated in *Figure 7* and is followed by explanations of the Pro API functions used to perform the tasks in the workflow.

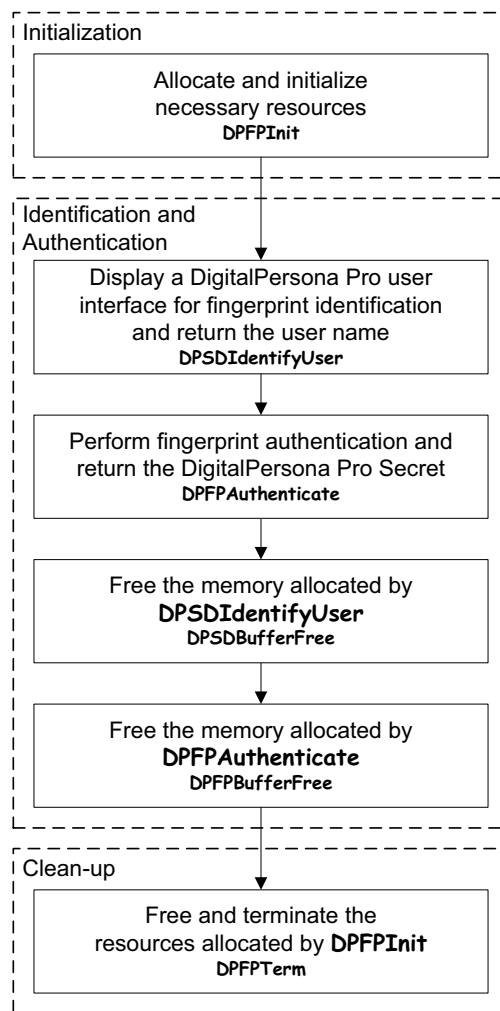


Figure 7. Typical fingerprint identification with fingerprint authentication workflow: With UI Support

Initialization

- Initialize and allocate necessary resources by calling the **DPFPInit** function (*page 46*).

Identification and Authentication

1. Display a DigitalPersona Pro user interface for fingerprint identification and return the user name and the handle to the supplied fingerprint credential by calling the **DPSDIdentifyUser** function (*page 58*).
The user name and the handle to the supplied fingerprint credential are passed to the **DPFPAuthenticate** function.
2. Perform fingerprint authentication and return the DigitalPersona Pro Secret by calling the **DPFPAuthenticate** function (*page 28*).
3. Free the memory allocated by calls to the **DPSDIdentifyUser** function by calling the **DPSDBufferFree** function (*page 53*).
4. Free the memory allocated by the **DPFPAuthenticate** function by calling the **DPFPBufferFree** function (*page 30*).

Clean-up

- Free and terminate the resources allocated by the **DPFPInit** function by calling the **DPFPTerm** function (*page 52*).

This chapter defines the functions, data structures, data types, and constants used in the Pro API. The information in this chapter is extracted from the `DPFPApi.h` and `DPSDApi.h` files located in the `<destination folder>\Include` folder.

Functions

This section defines the Pro API functions. Use the following list to quickly locate the functions contained in this section by name, by page, or by description.

| Function | Page | Description |
|--------------------------------------|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DPFPAddUserToIdentList | 28 | Adds a user to the kiosk identification list. |
| DPFPAuthenticate | 28 | Performs fingerprint authentication and returns the DigitalPersona Pro Secret. |
| DPFPBufferFree | 30 | Frees the memory previously allocated by the DPFPAuthenticate , DPFPEnumerateDevices , DPFPGetUserInfo , DPFPIdentify , and DPFPIdentifyUser functions. |
| DPFPCommitRegistrationChanges | 30 | Saves all changes made during a fingerprint enrollment operation. |
| DPFPCreateAcquisition | 31 | Creates a fingerprint acquisition operation. |
| DPFPCreateRegistration | 35 | Creates a fingerprint enrollment operation. |
| DPFPDeleteFinger | 38 | Deletes a stored fingerprint credential. |
| DPFPDeleteSecret | 39 | Deletes secret content from a user record. |
| DPFPDestroyAcquisition | 40 | Destroys a fingerprint acquisition operation previously created by the DPFPCreateAcquisition function and frees all resources associated with the operation. |
| DPFPDestroyRegistration | 40 | Destroys a fingerprint enrollment operation previously created by the DPFPCreateRegistration function and frees all resources associated with the operation. |
| DPFPEnumerateDevices | 41 | Enumerates available fingerprint readers. |
| DPFPGetDeviceInfo | 42 | Retrieves information about a fingerprint reader. |
| DPFPGetUserInfo | 42 | Retrieves information about a user. |
| DPFPGetVersion | 44 | Retrieves the DigitalPersona Pro API version information. |

| Function | Page | Description |
|------------------------------------|------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DPFPIdentify | 44 | Performs fingerprint identification and returns the user name. |
| DPFPIdentifyUser | 45 | Authenticates a user on the local machine. |
| DPFPInit | 46 | Allocates and initializes necessary resources. |
| DPFPisRegistrationAllowed | 47 | Checks if a user can be enrolled in the system. |
| DPFPisRegistrationAllowedEx | 48 | Checks if a user can be enrolled in the system using additional credentials. |
| DPFPStartAcquisition | 49 | Subscribes the fingerprint-enabled application (FEA) to receive fingerprint acquisition operation event notifications. |
| DPFPStartRegistration | 49 | Subscribes the FEA to receive fingerprint enrollment operation event notifications and creates a stored fingerprint credential. |
| DPFPStopAcquisition | 51 | Unsubscribes the FEA from receiving fingerprint acquisition operation event notifications. |
| DPFPStopRegistration | 51 | Unsubscribes the FEA from receiving fingerprint enrollment operation event notifications. |
| DPFPTerm | 52 | Frees and terminates the resources allocated by the DPFPInit function. |
| DPFPWriteSecretContent | 52 | Writes secret content to a user record. |
| Additional UI Support Functions | | |
| DPSDBufferFree | 53 | Frees the memory previously allocated by the DPSDIdentifyUser and DPSDIdentifyLogonUser functions. |
| DPSDIdentifyLogonUser | 54 | Displays a DigitalPersona Pro user interface for fingerprint identification and returns a token handle. |
| DPSDIdentifyUser | 55 | Displays a DigitalPersona Pro user interface for fingerprint identification and returns the user name and the handle to a supplied fingerprint credential. |
| DPSDVerifyLogonUser | 57 | Displays a DigitalPersona Pro user interface for fingerprint authentication and returns a token handle. |
| DPSDVerifyUser | 58 | Displays a DigitalPersona Pro user interface for fingerprint authentication and returns the handle to a supplied fingerprint credential. |

Use the following list to quickly locate the functions contained in this section by category, by name, by description, or by page.

| Function | Page | Description |
|---------------------------------------------------------------------|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Initialization and Termination Functions | | |
| DPFPInit | 46 | Allocates and initializes necessary resources. |
| DPFPTerm | 52 | Frees and terminates the resources allocated by the DPFPInit function. |
| Fingerprint Enrollment Operation Functions | | |
| DPFPCreateRegistration | 35 | Creates a fingerprint enrollment operation. |
| DPFPDestroyRegistration | 40 | Destroys a fingerprint enrollment operation previously created by the DPFPCreateRegistration function and frees all resources associated with the operation. |
| DPFPStartRegistration | 49 | Subscribes the FEA to receive fingerprint enrollment operation event notifications and creates a stored fingerprint credential. |
| DPFPStopRegistration | 51 | Unsubscribes the FEA from receiving fingerprint enrollment operation event notifications. |
| DPFPDeleteFinger | 38 | Deletes a stored fingerprint credential. |
| DPFPCommitRegistrationChanges | 30 | Saves all changes made during a fingerprint enrollment operation. |
| Fingerprint Acquisition Operation Functions | | |
| DPFPCreateAcquisition | 31 | Creates a fingerprint acquisition operation. |
| DPFPDestroyAcquisition | 40 | Destroys a fingerprint acquisition operation previously created by the DPFPCreateAcquisition function and frees all resources associated with the operation. |
| DPFPStartAcquisition | 49 | Subscribes the FEA to receive fingerprint acquisition operation event notifications. |
| DPFPStopAcquisition | 51 | Unsubscribes the FEA from receiving fingerprint acquisition operation event notifications. |
| Fingerprint Authentication and Fingerprint Identification Functions | | |
| DPFPAuthenticate | 28 | Performs fingerprint authentication and returns the DigitalPersona Pro Secret. |
| DPSDVerifyUser | 58 | Displays a DigitalPersona Pro user interface for fingerprint authentication and returns the handle to a supplied fingerprint credential. |

| Function | Page | Description |
|------------------------------------|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DPSDVerifyLogonUser | 57 | Displays a DigitalPersona Pro user interface for fingerprint authentication and returns a token handle. |
| DPFPIdentify | 44 | Performs fingerprint identification and returns the user name. |
| DPFPIdentifyUser | 45 | Authenticates a user on the local machine. |
| DPSDIdentifyUser | 55 | Displays a DigitalPersona Pro user interface for fingerprint identification and returns the user name and the handle to a supplied fingerprint credential. |
| DPDSIdentifyLogonUser | 54 | Displays a DigitalPersona Pro user interface for fingerprint identification and returns a token handle. |
| DPFPAddUserToIdentList | 28 | Adds a user to the kiosk identification list. |
| Memory Management Functions | | |
| DPFPBufferFree | 30 | Frees the memory previously allocated by the DPFPAuthenticate , DPFPEnumerateDevices , DPFPGetUserInfo , DPFPIdentify , and DPFPIdentifyUser functions. |
| DPSDBufferFree | 53 | Frees the memory previously allocated by the DPSDIdentifyUser and DPDSIdentifyLogonUser functions. |
| User Management Functions | | |
| DPFPGetUserInfo | 42 | Retrieves information about a user. |
| DPFPisRegistrationAllowed | 47 | Checks if a user can be enrolled in the system. |
| DPFPisRegistrationAllowedEx | 48 | Checks if a user can be enrolled in the system using additional credentials. |
| Device Management Functions | | |
| DPFPEnumerateDevices | 41 | Enumerates available fingerprint readers. |
| DPFPGetDeviceInfo | 42 | Retrieves information about a fingerprint reader. |
| Secret Management Functions | | |
| DPFPWriteSecretContent | 52 | Writes secret content to a user record. |
| DPFPDeleteSecret | 39 | Deletes secret content from a user record. |
| Version Information Functions | | |
| DPFPGetVersion | 44 | Retrieves the DigitalPersona Pro API version information. |

DPFPAddUserToIdentList

Adds a user to the kiosk identification list.

Syntax

```
DPFP_STDAPI DPFPAddUserToIdentList(
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE NameType,
    const DP_AUTHENTICATION_INFO* pAuthInfo
);
```

Parameter Names

| | |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------|
| szUserName | [in] Pointer to a null-terminated string containing the user name |
| NameType | [in] User name type. Valid values for this parameter are listed in Table 5 on page 43. |
| pAuthInfo | [in] Pointer to the credentials to be used. If this parameter is set to NULL , calling thread impersonation data is used. |

Return Values

| | |
|----------------|----------------------------------------------------------|
| S_OK | The user was added to the kiosk identification list. |
| S_FALSE | The user was not added to the kiosk identification list. |

Library

DPFPApi.dll

DPFPAuthenticate

Performs fingerprint authentication and returns the DigitalPersona Pro Secret. This function requires both a user name and the handle to a supplied fingerprint credential as input.

IMPORTANT: This function allocates memory, which *must* be freed by calling the **DPFPBufferFree** function (page 18).

Syntax

```

DPFP_STDAPI DPFPAuthenticate(
    HDPCREDENTIAL hCredential,
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE NameType,
    BOOL bForceRemote,
    LPCWSTR szSecretName,
    DATA_BLOB* pSecret
);

```

Parameter Names

| | |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hCredential | [in] Handle to the supplied fingerprint credential |
| szUserName | [in] Pointer to a null-terminated string containing the user name |
| NameType | [in] User name type. Valid values for this parameter are listed in Table 5 on page 43. |
| bForceRemote | [in] Boolean parameter that controls whether remote fingerprint authentication takes place. If the value of this parameter is TRUE and the server is accessible, fingerprint authentication takes place on the server. If the value is FALSE , fingerprint authentication takes place on the local machine. |
| szSecretName | [in] Pointer to a null-terminated string containing the name of the secret to be retrieved |
| pSecret | [out] Pointer to the structure of type DATA_BLOB to be filled with the secret content |

Return Values

| | |
|----------------|-------------------------------------------------------|
| S_OK | The secret content was returned by the server. |
| S_FALSE | The secret content was returned by the local machine. |

Library

DPFPApi.dll

See Also

DPFPStartAcquisition on page 49

DPFPStopAcquisition on page 51

DPFPBufferFree

Frees the memory previously allocated by the `DPFPAuthenticate`, `DPFPEnumerateDevices`, `DPFPGetUserInfo`, `DPFPIdentify`, and `DPFPIdentifyUser` functions.

Syntax

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

Parameter Names

| | |
|----------------|----------------------------------------|
| <code>p</code> | [in] Pointer to the memory to be freed |
|----------------|----------------------------------------|

Library

DPFPApi.dll

See Also

`DPFPAuthenticate` on page 28

`DPFPEnumerateDevices` on page 41

`DPFPGetUserInfo` on page 42

`DPFPIdentify` on page 44

`DPFPIdentifyUser` on page 45

DPFPCommitRegistrationChanges

Saves all of the changes made during a fingerprint enrollment operation, that is, this function adds a stored fingerprint credential(s) to a user record and deletes a stored fingerprint credential(s) from a user record.

IMPORTANT: All fingerprint enrollment operations *must* be completed by calling the `DPFPCommitRegistrationChanges` function.

Syntax

```
DPFP_STDAPI DPFPCommitRegistrationChanges(  
    HDPOPERATION hOperation,  
    const DP_AUTHENTICATION_INFO* pAuthInfo  
);
```

Parameter Names

| | |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------|
| hOperation | [in] Handle to the operation |
| pAuthInfo | [in] Pointer to the credentials to be used. If this parameter is set to NULL , calling thread impersonation data is used. |

Return Values

| | |
|-------------|-------------------------|
| S_OK | The changes were saved. |
|-------------|-------------------------|

Library

DPFPApi.dll

See Also

DPFPDeleteFinger on page 38

DPFPStartRegistration on page 49

DPFPStopRegistration on page 51

DPFPCreateAcquisition

Creates a fingerprint acquisition operation. During the creation of the operation, the fingerprint-enabled application (FEA) specifies the handle to the window to be notified of operation-related events as well as the window message to be sent as a notification. The **wParam** of the message specifies the event type, and the value of **lParam** (if used) is event-specific.

IMPORTANT: To free memory allocated for the fingerprint acquisition operation created, the **DPFPDestroyAcquisition** function (page 40) *must* be called for the handle returned in the **phOperation** parameter.

Syntax

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

Parameter Names

| | |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| eAcquisitionPriority | <p>[in] Fingerprint acquisition operation priority. Valid values for this parameter are</p> <p>DP_PRIORITY_HIGH. High priority. The subscriber uses this priority to acquire device events exclusively. Only one subscriber with this priority is allowed.</p> <p>DP_PRIORITY_NORMAL. 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.</p> <p>DP_PRIORITY_LOW. Low priority. The subscriber uses this priority to acquire device events only if there are no subscribers with high or normal priority. Only one subscriber with this priority is allowed.</p> |
| DevUID | [in] Serial number of a specific fingerprint reader. If any fingerprint reader can be used, set this parameter to the value GUID_NULL . |
| uSampleType | [in] The only supported value for this parameter is DP_SAMPLE_TYPE_VER . |
| hWnd | [in] Handle to the window to be notified of operation events |
| uMsg | [in] Windows message to be sent as an event notification. The event is encoded in wParam , while the meaning of the lParam (if used) depends on the meaning of wParam . Possible events are listed in <i>Table 3</i> below. |
| phOperation | [out] Pointer to the operation handle to be filled if the operation is created |

Table 3. **wParam** and **lParam** values for the **uMsg** parameter of the **DPFPCreateAcquisition** function

| wParam Value | Description | lParam Value |
|----------------------|-------------------------------------------------|--------------------------------------------------------------------------------------|
| WN_COMPLETED | A supplied fingerprint credential was acquired. | Contains the handle to a parameter of type HDPCREDENTIAL (<i>page 65</i>). |
| WN_DISCONNECT | The fingerprint reader was disconnected. | Contains a pointer to the device UID (see DP_DEVICE_INFO on <i>page 62</i>). |
| WN_ERROR | An error occurred. | Contains the error code that is returned. |

Table 3. wParam and lParam values for the uMsg parameter of the DPFPCreateAcquisition function (continued)

| wParam Value | Description | lParam Value |
|----------------------|--------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WN_FINGER_GONE | The finger was removed from the fingerprint reader. | |
| WN_FINGER_TOUCHED | The fingerprint reader was touched. | |
| WN_IMAGE_READY | A fingerprint image is ready for processing. | |
| WN_OPERATION_STOPPED | The operation was stopped by calling the DPFPStopAcquisition function. | |
| WN_RECONNECT | The fingerprint reader was reconnected. | Contains a pointer to the device UID (see DP_DEVICE_INFO on page 62). |
| WN_SAMPLE_QUALITY | Provides information about the quality of the <i>fingerprint image</i> . | <p>Contains the fingerprint image quality, which is represented by the following values:</p> <p>Area (touch) sensor image quality events</p> <p>DP_QUALITY_GOOD. The image is of good quality.</p> <p>DP_QUALITY_NONE. There is no image.</p> <p>DP_QUALITY_TOOLIGHT. The image is too light.</p> <p>DP_QUALITY_TOODARK. The image is too dark.</p> <p>DP_QUALITY_TOONOISY. The image is too noisy.</p> <p>DP_QUALITY_LOWCONTR. The image contrast is too low.</p> <p>DP_QUALITY_FTRNOTENOUGH. The image does not contain enough information.</p> |

Table 3. wParam and lParam values for the uMsg parameter of the DPFPCreateAcquisition function (continued)

| wParam Value | Description | lParam Value |
|--------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | | <p>DP_QUALITY_NOCENTRAL. The image is not centered.</p> |
| | | <p>Swipe sensor image quality events</p> |
| | | <p>DP_QUALITY_NOFINGER. The scanned object is not a finger.</p> <p>DP_QUALITY_TOOHIGH. The finger was too high on the swipe sensor.</p> <p>DP_QUALITY_TOLOW. The finger was too low on the swipe sensor.</p> <p>DP_QUALITY_TOLEFT. The finger was too close to left border of the swipe sensor.</p> <p>DP_QUALITY_TOORIGHT. The finger was too close to right border of the swipe sensor.</p> <p>DP_QUALITY_TOOSTRANGE. The scan looks strange.</p> <p>DP_QUALITY_TOOFAST. The finger was swiped too quickly.</p> <p>DP_QUALITY_TOOSKEWED. The image is too skewed.</p> <p>DP_QUALITY_TOOSHORT. The image is too short.</p> <p>DP_QUALITY_TOOSLOW. The finger was swiped too slowly.</p> |

Return Values

| | |
|-------------|----------------------------------------------------|
| S_OK | The fingerprint acquisition operation was created. |
|-------------|----------------------------------------------------|

Library

DPFPApi.dll

See Also

`DPFPDestroyAcquisition` on page 40

DPFPCreateRegistration

Creates a fingerprint enrollment operation. During the creation of the operation, the FEA specifies the handle to the window to be notified of operation-related events as well as the window message to be sent as a notification. The **wParam** of the message specifies the event type, and the value of **lParam** (if used) is event-specific.

IMPORTANT: To free memory allocated for the fingerprint enrollment operation created, the `DPFPDestroyRegistration` function (page 40) must be called for the handle returned in the **phOperation** parameter.

Syntax

```
DPFP_STDAPI DPFPCreateRegistration(
    REFGUID DevUID,
    HWND hWnd,
    ULONG uMsg,
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE NameType,
    HDPOPERATION* phOperation
);
```

Parameter Names

| | |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DevUID | [in] Serial number of a specific fingerprint reader. If any fingerprint reader can be used, set this parameter to the value <code>GUID_NULL</code> . |
| hWnd | [in] Handle to the window to be notified of operation events |
| uMsg | [in] Windows message to be sent as an event notification. The event is encoded in wParam , while the meaning of the lParam (if used) depends on the meaning of wParam . Possible events are listed in <i>Table 4</i> below. |
| szUserName | [in] Pointer to a null-terminated string containing the user name |
| NameType | [in] User name type. Valid values for this parameter are listed in <i>Table 5</i> on page 43. |
| phOperation | [out] Pointer to the operation handle to be filled if the operation is created |

Table 4. `wParam` and `lParam` values for the `uMsg` parameter of the `DPFPCreateRegistration` function

| wParam Value | Description | lParam Value |
|----------------------------------------|--------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>WN_DISCONNECT</code> | The fingerprint reader was disconnected. | Contains a pointer to the device UID (see <code>DP_DEVICE_INFO</code> on page 62). |
| <code>WN_ERROR</code> | An error occurred. | Contains the error code that is returned. |
| <code>WN_FEATURE_SET_ADDED</code> | A supplied fingerprint credential was added to the stored fingerprint credential. | |
| <code>WN_FINGER_GONE</code> | The finger was removed from the fingerprint reader. | |
| <code>WN_FINGER_TOUCHED</code> | The fingerprint reader was touched. | |
| <code>WN_IMAGE_READY</code> | A fingerprint image is ready for processing. | |
| <code>WN_OPERATION_STOPPED</code> | The operation was stopped by calling the <code>DPFPStopRegistration</code> function. | |
| <code>WN_RECONNECT</code> | The fingerprint reader was reconnected. | Contains a pointer to the device UID (see <code>DP_DEVICE_INFO</code> on page 62). |
| <code>WN_REGISTRATION_COMPLETED</code> | A fingerprint enrollment template was created. | |
| <code>WN_SAMPLE_QUALITY</code> | Provides information about the quality of the <i>fingerprint image</i> . | <p>Contains the fingerprint image quality, which is represented by the following values:</p> <p>Area (touch) sensor image quality events</p> <p><code>DP_QUALITY_GOOD</code>. The image is of good quality.</p> <p><code>DP_QUALITY_NONE</code>. There is no image.</p> <p><code>DP_QUALITY_TOOLIGHT</code>. The image is too light.</p> |

Table 4. wParam and lParam values for the uMsg parameter of the DPFPCreateRegistration function (continued)

| wParam Value | Description | lParam Value |
|--------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | | <p>DP_QUALITY_TOODARK. The image is too dark.</p> <p>DP_QUALITY_TOONOISY. The image is too noisy.</p> <p>DP_QUALITY_LOWCONTR. The image contrast is too low.</p> <p>DP_QUALITY_FTRNOTENOUGH. The image does not contain enough information.</p> <p>DP_QUALITY_NOCENTRAL. The image is not centered.</p> <hr/> <p>Swipe sensor quality events</p> <hr/> <p>DP_QUALITY_NOFINGER. The scanned object is not a finger.</p> <p>DP_QUALITY_TOOHIGH. The finger was too high on the swipe sensor.</p> <p>DP_QUALITY_TOOLOW. The finger was too low on the swipe sensor.</p> <p>DP_QUALITY_TOOLEFT. The finger was too close to left border of the swipe sensor.</p> <p>DP_QUALITY_TOORIGHT. The finger was too close to right border of the swipe sensor.</p> <p>DP_QUALITY_TOOSTRANGE. The scan looks strange.</p> <p>DP_QUALITY_TOOFAST. The finger was swiped too quickly.</p> <p>DP_QUALITY_TOOSKEWED. The image is too skewed.</p> |

Table 4. `wParam` and `lParam` values for the `uMsg` parameter of the `DPFPCreateRegistration` function (continued)

| <code>wParam</code> Value | Description | <code>lParam</code> Value |
|---------------------------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| | | <p><code>DP_QUALITY_TOOSHORT</code>. The image is too short.</p> <p><code>DP_QUALITY_TOOSLOW</code>. The finger was swiped too slowly.</p> |

Return Values

| | |
|-------------------|---------------------------------------------------|
| <code>S_OK</code> | The fingerprint enrollment operation was created. |
|-------------------|---------------------------------------------------|

Library

DPFPApi.dll

See Also

`DPFPCommitRegistrationChanges` on page 30

`DPFPDestroyRegistration` on page 40

DPFPDeleteFinger

Deletes a stored fingerprint credential.

IMPORTANT: This operation *must* be completed by calling the `DPFPCommitRegistrationChanges` function (page 30).

Syntax

```
DPFP_STDAPI DPFPDeleteFinger(
    HDPOPERATION hOperation,
    ULONG uFinger
);
```

Parameter Names

| | |
|-------------------------|-----------------------------------------------------------------------------------------------------|
| <code>hOperation</code> | [in] Handle to the operation |
| <code>uFinger</code> | [in] Finger number to be deleted. Valid values for this parameter are listed in Table 6 on page 50. |

Return Values

| | |
|-------------|------------------------------------------------|
| S_OK | The stored fingerprint credential was deleted. |
|-------------|------------------------------------------------|

Library

DPFPApi.dll

See Also

DPFPCommitRegistrationChanges on *page 30*

DPFPDeleteSecret

Deletes secret content from a user record.

Syntax

```
DPFP_STDAPI DPFPDeleteSecret(
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE NameType,
    LPCWSTR szSecretName,
    const DP_AUTHENTICATION_INFO* pAuthInfo
);
```

Parameter Names

| | |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------|
| szUserName | [in] Pointer to a null-terminated string containing the user name |
| NameType | [in] User name type. Valid values for this parameter are listed in Table 5 on <i>page 43</i> . |
| szSecretName | [in] Pointer to a null-terminated string containing the name of the secret to be retrieved |
| pAuthInfo | [in] Pointer to the credentials to be used. If this parameter is set to NULL , calling thread impersonation data is used. |

Return Values

| | |
|-------------|---------------------------------|
| S_OK | The secret content was deleted. |
|-------------|---------------------------------|

Library

DPFPApi.dll

See Also

`DPFPWriteSecretContent` on page 52

DPFPDestroyAcquisition

Destroys a fingerprint acquisition operation previously created by the `DPFPCreateAcquisition` function (page 31) and frees all resources associated with the operation.

Syntax

```
DPFP_STDAPI DPFPDestroyAcquisition(  
    HDPOPERATION hOperation  
);
```

Parameter Names

| | |
|-------------------------|----------------------------------------------|
| <code>hOperation</code> | [in] Handle to the operation to be destroyed |
|-------------------------|----------------------------------------------|

Return Values

| | |
|-------------------|------------------------------------------------------|
| <code>S_OK</code> | The fingerprint acquisition operation was destroyed. |
|-------------------|------------------------------------------------------|

Library

DPFPApi.dll

See Also

`DPFPCreateAcquisition` on page 31

DPFPDestroyRegistration

Destroys a fingerprint enrollment operation previously created by the `DPFPCreateRegistration` function (page 35) and frees all resources associated with the operation.

Syntax

```
DPFP_STDAPI DPFPDestroyRegistration(  
    HDPOPERATION hOperation  
);
```

Parameter Names

| | |
|-------------------|----------------------------------------------|
| hOperation | [in] Handle to the operation to be destroyed |
|-------------------|----------------------------------------------|

Return Values

| | |
|-------------|-----------------------------------------------------|
| S_OK | The fingerprint enrollment operation was destroyed. |
|-------------|-----------------------------------------------------|

Library

DPFPApi.dll

See Also

DPFPCommitRegistrationChanges on page 30

DPFPCreateRegistration on page 35

DPFPDeleteFinger on page 38

DPFPEnumerateDevices

Enumerates available fingerprint readers.

IMPORTANT: This function allocates memory, which *must* be freed by calling the **DPFPBufferFree** function (page 18).

Syntax

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

Parameter Names

| | |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| puDevCount | [out] Pointer to the number of available fingerprint readers. If no fingerprint readers are found, the value of this parameter is 0. |
| ppDevUID | [out] Pointer to be filled with the pointer to the array of serial numbers of available fingerprint readers. If the value of this parameter is NULL , only the number of available fingerprint readers is returned. |

Return Values

| | |
|-------------------|-------------------------|
| <code>S_OK</code> | The function succeeded. |
|-------------------|-------------------------|

Library

DPFPApi.dll

DPFPGetDeviceInfo

Retrieves information about a fingerprint reader and returns it in the structure of type `DP_DEVICE_INFO` (page 62).

Syntax

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

Parameter Names

| | |
|-----------------------|------------------------------------------------|
| <code>DevUID</code> | [in] Fingerprint reader serial number |
| <code>pDevInfo</code> | [in/out] Pointer to the structure to be filled |

Return Values

| | |
|-------------------|---------------------------------------------------|
| <code>S_OK</code> | The fingerprint reader information was retrieved. |
|-------------------|---------------------------------------------------|

Library

DPFPApi.dll

See Also

`DPFPEnumerateDevices` on page 41

DPFPGetUserInfo

Retrieves information about the a user and returns it in the structure of type `DP_USER_INFO_0` (page 64).

IMPORTANT: This function allocates memory, which *must* be freed by calling the `DPFPBufferFree` function (page 18).

Syntax

```

DPFP_STDAPI DPFPGetUserInfo (
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE NameType,
    UINT Level,
    PVOID* ppInfo
);

```

Parameter Names

| | |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| szUserName | [in] Pointer to a null-terminated string containing the user name |
| NameType | [in] User name type. Valid values for this parameter are listed in <i>Table 5</i> below. |
| Level | [in] Information level. This parameter should be set to 0. |
| ppInfo | [out] Pointer to the pointer to the user information returned. When the parameter Level is set to 0, the ppInfo parameter points to the structure of type DP_USER_INFO_0 . |

Table 5. Values for parameters of type **DP_USER_NAME_TYPE**

| Value | Description |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| DP_USER_NAME_ACCOUNT_SIMPLE | The account name format used in Microsoft® Windows NT® 4.0, for example, "jdoe" |
| DP_USER_NAME_DISPLAY | A friendly display name, for example, "John Doe" |
| DP_USER_NAME_DNS_DOMAIN | A DNS domain name, for example, "thecompany.com" |
| DP_USER_NAME_NET_BIOS_DOMAIN | NetBIOS domain name, for example, "THE_COMPANY" |
| DP_USER_NAME_SAM_COMPATIBLE | A Microsoft Windows NT 4.0 account name, for example, "the_company\jdoe" (only the domain name should end with a back slash) |
| DP_USER_NAME_SID | A user SID string, for example, "S-1-5-21-1004" |
| DP_USER_NAME_UNIQUE_ID | A GUID string returned by the IIDFromString function, for example, "{4fa050f0-f561-11cf-bdd9-00aa003a77b6}" |
| DP_USER_NAME_UNKNOWN | A name not associated with any Windows account. This value is to be used for local databases only. |
| DP_USER_NAME_USER_PRINCIPAL | A user principal name, for example, "jdoe@thecompany.com" |

Return Values

| | |
|----------------------|---------------------------------------------------|
| <code>S_OK</code> | User information was stored on the remote server. |
| <code>S_FALSE</code> | User information was stored on the local machine. |

Library

DPFPApi.dll

DPFPGetVersion

Retrieves the Pro API version information and returns it in the structure of type `DP_PRODUCT_VERSION` (page 64).

Syntax

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

Parameter Names

| | |
|-----------------------|----------------------------------------------------------------------|
| <code>pVersion</code> | [out] Pointer to the structure receiving the API version information |
|-----------------------|----------------------------------------------------------------------|

Return Values

| | |
|-------------------|------------------------------------------------|
| <code>S_OK</code> | The Pro API version information was retrieved. |
|-------------------|------------------------------------------------|

Library

DPFPApi.dll

DPFPIdentify

Performs fingerprint identification and returns the user name. This function requires the handle to a supplied fingerprint credential as input.

IMPORTANT: This function allocates memory, which *must* be freed by calling the `DPFPBufferFree` function (page 18).

Syntax

```
DPFP_STDAPI DPFPIdentify(
    HDPCREDENTIAL hCredential,
    LPWSTR* ppUserName
);
```

Parameter Names

| | |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hCredential | [in] Handle to the supplied fingerprint credential |
| ppUserName | [out] Pointer to be filled with the pointer to the name of the identified user. Names are returned in DP_USER_NAME_SAM_COMPATIBLE format (see Table 5 on page 43). |

Return Values

| | |
|----------------|-----------------------------|
| S_OK | The user is identified. |
| S_FALSE | The user is not identified. |

Library

DPFPApi.dll

See Also

DPFPStartAcquisition on page 49

DPFPDestroyAcquisition on page 40

DPFPIdentifyUser

Authenticates a user on the local machine. You can call this function to quickly verify that the supplied fingerprint credential is from the specified user. This function requires both a user name and the handle to a supplied fingerprint credential as input.

IMPORTANT: This function allocates memory, which *must* be freed by calling the **DPFPBufferFree** function (page 18).

Syntax

```
DPFP_STDAPI DPFPIdentifyUser(
    HDPCREDENTIAL hCredential,
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE NameType
);
```

Parameter Names

| | |
|--------------------|----------------------------------------------------------------------------------------|
| hCredential | [in] Handle to the supplied fingerprint credential |
| szUserName | [in] Pointer to a null-terminated string containing the user name |
| NameType | [in] User name type. Valid values for this parameter are listed in Table 5 on page 43. |

Return Values

| | |
|----------------|-----------------------------|
| S_OK | The user is identified. |
| S_FALSE | The user is not identified. |

Library

DPFPApi.dll

See Also

DPFPStartAcquisition on page 49

DPFPDestroyAcquisition on page 40

DPFPInit

Allocates and initializes necessary resources.

IMPORTANT: The **DPFPInit** function *must* be called before calling any other functions except the **DPFPBufferFree** function (page 30), and every successful call to this function *must* be paired with a call to the **DPFPTerm** function (page 52).

Syntax

```
DPFP_STDAPI DPFPInit();
```

Return Values

| | |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| S_OK | The function succeeded. |
| S_FALSE | The library is already initialized. |
| 0x800706B3 | The following standard error is returned: "The RPC server is not listening." This means that the Biometric Authentication Service has not started. |

Library

DPFPapi.dll

See Also

DPFPTerm on page 52

DPFPisRegistrationAllowed

Checks if a user can be enrolled in the system.

Syntax

```
DPFP_STDAPI DPFPisRegistrationAllowed(
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE NameType
);
```

Parameter Names

| | |
|-------------------|----------------------------------------------------------------------------------------|
| szUserName | [in] Pointer to a null-terminated string containing the user name |
| NameType | [in] User name type. Valid values for this parameter are listed in Table 5 on page 43. |

Return Values

| | |
|----------------|--------------------------------------------|
| S_OK | The user can be enrolled in the system. |
| S_FALSE | The user cannot be enrolled in the system. |

Library

DPFPapi.dll

See Also

`DPFPisRegistrationAllowedEx` on page 48

DPFPisRegistrationAllowedEx

Checks if a user can be enrolled in the system. This function is used when additional credentials are required from the enrollee or from another person, as in the case of attended enrollment. The `DPFPisRegistrationAllowedEx` function is also used for Pro Kiosk when the enrollee is not the logon user.

Syntax

```
DPFP_STDAPI_(HRESULT) DPFPisRegistrationAllowedEx(
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE NameType,
    const DP_AUTHENTICATION_INFO* pAuthInfo
);
```

Parameter Names

| | |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------|
| szUserName | [in] Pointer to a null-terminated string containing the user name |
| NameType | [in] User name type. Valid values for this parameter are listed in Table 5 on page 43. |
| pAuthInfo | [in] Pointer to the credentials to be used. If this parameter is set to NULL , calling thread impersonation data is used. |

Return Values

| | |
|----------------|--------------------------------------------|
| S_OK | The user can be enrolled in the system. |
| S_FALSE | The user cannot be enrolled in the system. |

Library

DPFPapi.dll

See Also

`DPFPisRegistrationAllowed` on page 47

DPFPStartAcquisition

Subscribes the FEA to receive fingerprint acquisition operation event notifications. Before the FEA can receive messages, it must subscribe to them by calling the **DPFPStartAcquisition** function. The message defined in the **uMsg** parameter of the **DPFPCreateAcquisition** function (*page 19*) is sent to the FEA. Possible events are listed in Table 3 on *page 32*. Each process can have no more than one active subscription for each operation priority level.

IMPORTANT: Every successful call to the **DPFPStartAcquisition** function *must* be paired with a call to the **DPFPStopAcquisition** function (*page 51*).

Syntax

```
DPFP_STDAPI DPFPStartAcquisition(
    HDPOPERATION hOperation
);
```

Parameter Names

| | |
|-------------------|------------------------------|
| hOperation | [in] Handle to the operation |
|-------------------|------------------------------|

Return Values

| | |
|-------------|-----------------------------------------------------------------|
| S_OK | The FEA is subscribed to receive operation event notifications. |
|-------------|-----------------------------------------------------------------|

Library

DPFPApi.dll

See Also

DPFPStopAcquisition on *page 51*

DPFPStartRegistration

Subscribes the FEA to receive fingerprint enrollment operation event notifications and creates a stored fingerprint credential(s) to be used for the purpose of enrollment. Before the FEA can receive messages, it must subscribe to them by calling the **DPFPStartRegistration** function. The message defined in the **uMsg** parameter of the **DPFPCreateRegistration** function (*page 19*) is sent to the FEA. Possible events are listed in Table 4 on *page 36*.

Each process can have no more than one active subscription. The level of the fingerprint enrollment operation is always equal to the value **DP_PRIORITY_NORMAL**. Sequential calls to the **DPFPStartRegistration** function are permitted for the same handle to the operation and different fingers. Each subsequent call stops the previous fingerprint enrollment operation and starts a new one.

IMPORTANT: Every successful call to the **DPFPStartRegistration** function *must* be paired with a call to the **DPFPStopRegistration** function (*page 51*). In addition, to complete a fingerprint enrollment operation, you *must* call the **DPFPCommitRegistrationChanges** function (*page 18*).

Syntax

```
DPFP_STDAPI DPFPStartRegistration(
    HDPOPERATION hOperation,
    ULONG uFinger
);
```

Parameter Names

| | |
|-------------------|--------------------------------------------------------------------------------------------------------|
| hOperation | [in] Handle to the operation |
| uFinger | [in] Finger number to be enrolled. Valid values for this parameter are listed in <i>Table 6</i> below. |

Table 6. Values for the **uFinger** parameter

| Finger | Value | Finger | Value |
|--------------------|-------|---------------------|-------|
| Left little finger | 0 | Right thumb | 5 |
| Left ring finger | 1 | Right index finger | 6 |
| Left middle finger | 2 | Right middle finger | 7 |
| Left index finger | 3 | Right index finger | 8 |
| Left thumb | 4 | Right little finger | 9 |

Return Values

| | |
|-------------|-----------------------------------------------------------------|
| S_OK | The FEA is subscribed to receive operation event notifications. |
|-------------|-----------------------------------------------------------------|

Library

DPFPApi.dll

See Also

`DPFPCommitRegistrationChanges` on page 30

`DPFPDeleteFinger` on page 38

`DPFPStopRegistration` on page 51

DPFPStopAcquisition

Unsubscribes the FEA from receiving fingerprint acquisition operation event notifications.

Syntax

```
DPFP_STDAPI DPFPStopAcquisition(  
    HDPOPERATION hOperation  
);
```

Parameter Names

| | |
|-------------------------|------------------------------|
| <code>hOperation</code> | [in] Handle to the operation |
|-------------------------|------------------------------|

Return Values

| | |
|-------------------|-----------------------------------------------------------------------------|
| <code>S_OK</code> | The FEA has been unsubscribed from receiving operation event notifications. |
|-------------------|-----------------------------------------------------------------------------|

Library

DPFPApi.dll

See Also

`DPFPStartAcquisition` on page 49

DPFPStopRegistration

Unsubscribes the FEA from receiving fingerprint enrollment operation event notifications.

Syntax

```
DPFP_STDAPI DPFPStopRegistration(  
    HDPOPERATION hOperation  
);
```

Parameter Names

| | |
|-------------------|------------------------------|
| hOperation | [in] Handle to the operation |
|-------------------|------------------------------|

Return Values

| | |
|-------------|--------------------------------------------------------------------|
| S_OK | The FEA has been unsubscribed from receiving events notifications. |
|-------------|--------------------------------------------------------------------|

Library

DPFPApi.dll

See Also

[DPFPStartRegistration](#) on *page 49*

DPFPTerm

Frees and terminates the resources allocated by the `DPFPInit` function (*page 46*).

Syntax

```
DPFP_STDAPI_(void) DPFPTerm();
```

Library

DPFPApi.dll

See Also

[DPFPInit](#) on *page 46*

DPFPWriteSecretContent

Writes secret content to a user record.

Syntax

```
DPFP_STDAPI DPFPWriteSecretContent(
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE NameType,
    LPCWSTR szSecretName,
    const DATA_BLOB* pSecret,
    const DP_AUTHENTICATION_INFO* pAuthInfo
);
```

Parameter Names

| | |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------|
| szUserName | [in] Pointer to a null-terminated string containing the user name |
| NameType | [in] User name type. Valid values for this parameter are listed in Table 5 on page 43. |
| szSecretName | [in] Pointer to a null-terminated string containing the name of the secret to be retrieved |
| pSecret | [in] Pointer to the structure of type DATA_BLOB containing the secret data |
| pAuthInfo | [in] Pointer to the credentials to be used. If this parameter is set to NULL , calling thread impersonation data is used. |

Return Values

| | |
|----------------|-----------------------------------------------------|
| S_OK | The secret content was stored on the remote server. |
| S_FALSE | The secret content was stored on the local machine. |

Library

DPFPApi.dll

See Also

DPFPDeleteSecret on page 39

DPSDBufferFree

Frees the memory previously allocated by the **DPSDIdentifyUser** and **DPSDIdentifyLogonUser** functions.

Syntax

```
DPSD_STDAPI_(void) DPSDBufferFree(PVOID pBuffer);
```

Parameter Names

| | |
|----------------|----------------------------------------|
| pBuffer | [in] Pointer to the memory to be freed |
|----------------|----------------------------------------|

Library

DPSDApi.dll

See Also

DPDSDIdentifyLogonUser on page 54

DPDSDIdentifyUser on page 55

DPDSDIdentifyLogonUser

Displays a DigitalPersona Pro user interface for fingerprint identification with authentication. This function requires that the user provide a fingerprint. The **DPDSDIdentifyLogonUser** function is similar to the **DPDSDIdentifyUser** function (page 55). However, instead of returning the user name, the function retrieves the DigitalPersona Pro Secret, which is the user's logon password, from the DigitalPersona database. If the function succeeds, it returns a handle to a token that represents the logged-on user, performs user logon, and closes the dialog box. The token handle can then be used to impersonate the user or to create a process that runs in the context of the user. The dialog box can be modified by changing its title and the default text.

IMPORTANT: This function allocates memory, which *must* be freed by calling the **DPSDBufferFree** function (page 41).

Syntax

```
DPDSD_STDAPIDDPDSDIdentifyLogonUser(  
    HWND hParentWnd,  
    LPCWSTR szCaption,  
    LPCWSTR szText,  
    LPWSTR* pszUserName,  
    HANDLE* phToken  
);
```

Parameter Names

| | |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hParentWnd | [in] Handle to the parent window. This parameter can be any valid handle to a window, or it can be NULL if the dialog box has no parent. |
| szCaption | [in] Pointer to a null-terminated string that specifies the dialog box title. If this parameter is set to NULL , the string "Verify Your Identity" is displayed. |
| szText | [in] Pointer to a null-terminated string that specifies the dialog box text. If this parameter is set to NULL , the following default text is displayed: If Pro Kiosk is installed: "To access your account data, touch the fingerprint reader with any enrolled finger. If you have never used this kiosk or have not used it recently, you may need to provide your user name and domain name in addition to the fingerprint." If Pro Workstation is installed: "To access your account data, touch the fingerprint reader with any enrolled finger. You may need to provide your user name and domain name in addition to the fingerprint." |
| pszUserName | [out] Pointer to the address of the buffer receiving the pointer to a null-terminated string, which is the name of the identified user returned in DP_USER_NAME_SAM_COMPATIBLE format (see Table 5 on page 43). |
| phToken | [out] Pointer to the handle receiving a handle to the token that represents the specified user |

Return Values

| | |
|--------------------------------------------------|-------------------------------------------------------------------------------------|
| S_OK | The function succeeded. |
| 0x800704c7 HR (ERROR_CANCELLED) | The following standard error is returned: "The operation was canceled by the user." |

Library

DPSDApi.dll

DPSDIdentifyUser

Displays a DigitalPersona Pro user interface for fingerprint identification. This function requires that the user provide a fingerprint. If the function succeeds, it returns the user name and the handle to a supplied fingerprint credential. The dialog box can be modified by changing its title and the default text.

IMPORTANT: This function allocates memory, which *must* be freed by calling the **DPSDBufferFree** function (page 41).

Syntax

```
DPSD_STDAPIDPSDIdentifyUser (
    HWND hParentWnd,
    LPCWSTR szCaption,
    LPCWSTR szText,
    LPWSTR* pszUserName,
    HDPCREDENTIAL* phCredential
);
```

Parameter Names

| | |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hParentWnd | [in] Handle to the parent window. This parameter can be any valid handle to a window, or it can be NULL if the dialog box has no parent. |
| szCaption | [in] Pointer to a null-terminated string that specifies the dialog box title. If this parameter is set to NULL , the string "Verify Your Identity" is displayed. |
| szText | [in] Pointer to a null-terminated string that specifies dialog box text. If this parameter is set to NULL , the following default text is displayed: If Pro Kiosk is installed: "To access your account data, touch the fingerprint reader with any enrolled finger. If you have never used this kiosk or have not used it recently, you may need to provide your user name and domain name in addition to the fingerprint." If Pro Workstation is installed: "To access your account data, touch the fingerprint reader with any enrolled finger. You may need to provide your user name and domain name in addition to the fingerprint." |
| pszUserName | [out] Pointer to the address of the buffer receiving the pointer to a null-terminated string, which is the name of the identified user returned in DP_USER_NAME_SAM_COMPATIBLE format (see Table 5 on page 43). |
| phCredential | [out] Pointer to the handle receiving the handle to the supplied fingerprint credential |

Return Values

| | |
|--------------------------------------------|-------------------------------------------------------------------------------------|
| S_OK | The function succeeded. |
| 0x800704c7 HR (ERROR_CANCELLED) | The following standard error is returned: "The operation was canceled by the user." |

Library

DPSDApi.dll

DPSDVerifyLogonUser

Displays a DigitalPersona Pro user interface for fingerprint authentication. This function requires that the user provide a user name and a fingerprint. The **DPSDVerifyLogonUser** function is similar to the **DPSDVerifyUser** function (*page 58*). However, instead of returning the handle to the supplied fingerprint credential, the function retrieves the DigitalPersona Pro Secret, which is the user's logon password, from the DigitalPersona database. If the function succeeds, it returns a handle to a token that represents the logged-on user, performs user logon, and closes the dialog box. The token handle can then be used to impersonate the user or to create a process that runs in the context of the user. The dialog box can be modified by changing its title and the default text.

Syntax

```
DPSD_STDAPIDPSDVerifyLogonUser (
    HWND hParentWnd,
    LPCWSTR szCaption,
    LPCWSTR szText,
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE UserNameType,
    HANDLE* phToken
);
```

Parameter Names

| | |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hParentWnd | [in] Handle to the parent window. This parameter can be any valid handle to a window, or it can be NULL if the dialog box has no parent. |
| szCaption | [in] Pointer to a null-terminated string that specifies the dialog box title. If this parameter is set to NULL , the string "Verify Your Identity" is displayed. |
| szText | [in] Pointer to a null-terminated string that specifies dialog box text. If this parameter is set to NULL , the following default text is displayed: "To access your account data, touch the fingerprint reader with any enrolled finger." |
| szUserName | [in] Pointer to a null-terminated string that specifies the name of the user to be authenticated. If this parameter is set to NULL , the user who is currently logged on is authenticated. |
| UserNameType | [in] User name type. Valid values for this parameter are listed in Table 5 on <i>page 43</i> . However, this parameter cannot be set to the value DP_USER_NAME_UNKNOWN and is ignored if the szUserName parameter is set to NULL . |
| phToken | [out] Pointer to the handle receiving a handle to the token that represents the specified user |

Return Values

| | |
|----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>S_OK</code> | The function succeeded. |
| <code>0x8009030E</code> <code>SEC_E_NO_CREDENTIALS</code> | The following standard error is returned: "No credentials are available in the security package." Corrective action: The specified user must enroll fingerprints. |
| <code>0x80070525</code> <code>HR(ERROR_NO_SUCH_USER)</code> | The following standard error is returned: "The specified user does not exist." Corrective action: The <code>szUserName</code> parameter might be invalid. |
| <code>0x80070533</code> <code>HR(ERROR_ACCOUNT_DISABLED)</code> | The following standard error is returned: "Logon failure: account currently disabled." Corrective action: Contact your system administrator. |
| <code>0x80070775</code> <code>HR(ERROR_ACCOUNT_LOCKED_OUT)</code> | The following standard error is returned: "The referenced account is currently locked out and may not be logged on to." Corrective action: Contact your system administrator. |
| <code>0x800704c7</code> <code>HR(ERROR_CANCELLED)</code> | The following standard error is returned: "The operation was canceled by the user." |

Library

DPSDApi.dll

DPSDVerifyUser

Displays a DigitalPersona Pro user interface for fingerprint authentication. This function requires that the user provide a user name and a fingerprint. If the function succeeds, it returns the handle to a supplied fingerprint credential. The dialog box can be modified by changing its title and the default text.

Syntax

```
DPSD_STDAPIDPSDVerifyUser (
    HWND hParentWnd,
    LPCWSTR szCaption,
    LPCWSTR szText,
    LPCWSTR szUserName,
    DP_USER_NAME_TYPE UserNameType,
    HDPCREDENTIAL* phCredential
);
```

Parameter Names

| | |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hParentWnd | [in] Handle to the parent window. This parameter can be any valid handle to a window, or it can be NULL if the dialog box has no parent. |
| szCaption | [in] Pointer to a null-terminated string that specifies the dialog box title. If this parameter is set to NULL , the string "Verify Your Identity" is displayed. |
| szText | [in] Pointer to a null-terminated string that specifies dialog box text. If this parameter is set to NULL , the following default text is displayed: "To access your account data, touch the fingerprint reader with any enrolled finger." |
| szUserName | [in] Pointer to a null-terminated string that specifies the name of the user to be authenticated. If this parameter is set to NULL , the user who is currently logged on is authenticated. |
| UserNameType | [in] User name type. Valid values for this parameter are listed in Table 5 on page 43. However, this parameter cannot be set to DP_USER_NAME_UNKNOWN and is ignored if the szUserName parameter is set to NULL . |
| phCredential | [out] Pointer to the handle receiving the handle to the supplied fingerprint credential |

Return Values

| | |
|----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| S_OK | The function succeeded. |
| 0x8009030E SEC_E_NO_CREDENTIALS | The following standard error is returned: "No credentials are available in the security package." Corrective action: The specified user must enroll fingerprints. |
| 0x80070525 HR(ERROR_NO_SUCH_USER) | The following standard error is returned: "The specified user does not exist." Corrective action: The szUserName parameter might be invalid. |
| 0x80070533 HR(ERROR_ACCOUNT_DISABLED) | The following standard error is returned: "Logon failure: account currently disabled." Corrective action: Contact your system administrator. |
| 0x80070775 HR(ERROR_ACCOUNT_LOCKED_OUT) | The following standard error is returned: "The referenced account is currently locked out and may not be logged on to." Corrective action: Contact your system administrator. |
| 0x800704c7 HR(ERROR_CANCELLED) | The following standard error is returned: "The operation was canceled by the user." |

Library

DPSDApi.dll

Data Structures

This section defines the Pro API data structures.

DP_AUTHENTICATION_INFO

Credential structure.

Data Fields

| | |
|------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LPWSTR UserName | String containing the user name. |
| DP_USER_NAME_TYPE UserNameType | Indicates the user name type, as represented by the values listed in Table 5 on <i>page 43</i> . |
| DP_CREDENTIAL_TYPE CredentialType | Indicates the credential type, as represented by the following values: DP_CREDENTIAL_TYPE_PASSWORD. Clear text password. DP_CREDENTIAL_TYPE_FP_HANDLE. Handle returned by the DigitalPersona Pro system to the fingerprint credentials. DP_CREDENTIAL_TYPE_FP_DATA. Binary fingerprint data. |
| union | Credential data. |
| LPWSTR Password | Clear text password. |
| ULONG CredentialHandle | Handle returned by the DigitalPersona Pro system to the handle to the fingerprint credentials. |
| DATA_BLOB FingerprintData | Binary fingerprint data. |

DP_DEVICE_INFO

Device information structure.

Data Fields

| | |
|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GUID DeviceUid | Device UID (unique identifier) |
| DP_DEVICE_UID_TYPE eUidType | Indicates whether the device UID persists after reboot, as represented by the following values: DP_PERSISTENT_DEVICE_UID. Persistent UIDs. This value is hardware-dependent. DP_VOLATILE_DEVICE_UID. Volatile UIDs. This value is software-generated. |
| DP_DEVICE_MODALITY eDeviceModality | Indicates the modality that the device uses to capture fingerprint images, as represented by the following values: DP_AREA_DEVICE. Area (touch) sensor devices. DP_SWIPE_DEVICE. Swipe devices. DP_UNKNOWN_DEVICE_MODALITY. Devices for which the modality is not known. DP_DEVICE_MODALITY_NUM. Count of the defined modalities. |
| DP_DEVICE_TECHNOLOGY eDeviceTech | Indicates the device technology, as represented by the following values: DP_OPTICAL_DEVICE. Optical devices. DP_CAPACITIVE_DEVICE. Capacitive devices. DP_THERMAL_DEVICE. Thermal devices. DP_PRESSURE_DEVICE. Pressure devices. DP_UNKNOWN_DEVICE_TECHNOLOGY. Devices for which the technology is not known. DP_DEVICE_TECHNOLOGY_NUM. Count of the defined technologies. |
| DP_HW_INFO HwInfo | Device information in the structure of type DP_HW_INFO (page 63). |

DP_DEVICE_VERSION

Device hardware and firmware version number structure.

Data Fields

| | |
|---------------------------|----------------------|
| <code>ULONG uMajor</code> | Major version number |
| <code>ULONG uMinor</code> | Minor version number |
| <code>ULONG uBuild</code> | Build number |

DP_HW_INFO

Device hardware information structure.

Data Fields

| | |
|---------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>unsigned int uLanguageId</code> | Device language |
| <code>wchar_t szVendor[DP_MAX_USB_STRING_SIZE]</code> | Pointer to a null-terminated string containing the manufacturer name, for example, "Digital Persona, Inc." |
| <code>wchar_t szProduct[DP_MAX_USB_STRING_SIZE]</code> | Pointer to a null-terminated string containing the product name, for example, "U.are.U 4000B" |
| <code>wchar_t szSerialNb[DP_MAX_USB_STRING_SIZE]</code> | Pointer to a null-terminated string containing the serial number, for example, "{7C265680-0056-FFFF-680D-A74033B09615}" |
| <code>DP_DEVICE_VERSION HardwareRevision</code> | Hardware revision |
| <code>DP_DEVICE_VERSION FirmwareRevision</code> | Firmware revision |

DP_PRODUCT_VERSION

DigitalPersona product version structure.

Data Fields

| | |
|-----------------------|-------------------------|
| WORD wMajor | Major product number |
| WORD wMinor | Minor product number |
| WORD wRevision | Revision product number |
| WORD wBuild | Build number |

DP_USER_INFO_0

Level 0 user information structure.

Data Fields

| | |
|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GUID usri0_unique_id | User's GUID |
| ULONG usri0_flags | User's account control bits |
| ULONG usri0_fp_credential_mask | Mask that is a combination of the values representing a user's enrolled fingers. For example, if a user's right index finger and right middle finger are enrolled, the value of this field is 00000000 01100000. All possible values for the usri0_fp_credential_mask field are listed in <i>Table 7</i> below. |
| ULONG usri0_fp_max_allowed | Maximum number of fingerprints allowed |

Table 7. Values for the **usri0_fp_credential_mask** field

| Finger | Binary Representation | Finger | Binary Representation |
|--------------------|-----------------------|---------------------|-----------------------|
| Left little finger | 00000000 00000001 | Right thumb | 00000000 00010000 |
| Left ring finger | 00000000 00000010 | Right index finger | 00000000 00100000 |
| Left middle finger | 00000000 00000100 | Right middle finger | 00000000 01000000 |
| Left index finger | 00000000 00001000 | Right index finger | 00000000 10000000 |
| Left thumb | 00000000 00001000 | Right little finger | 00000001 00000000 |

Data Types and Constants

This section identifies some Pro API types and constants.

HDPOPERATION

Operation handle.

```
typedef unsigned long HDPOPERATION;
```

HDPCREDENTIAL

Credential handle. Currently, the only valid credentials are fingerprint credentials.

```
typedef unsigned long HDPCREDENTIAL;
```

DP_MAX_USB_STRING_SIZE

The maximum string length supported by USB devices, which is 128.

```
#define DP_MAX_USB_STRING_SIZE (128)
```

This comparative glossary is for those who are already familiar with the terminology and concepts presented in the *DigitalPersona Pro for Active Directory Administrator Guide*. Some of the terms were changed to make fingerprint authentication easier to understand and to conform to current biometrics industry standards. The following table lists the original terms used in the administrator guide, the corresponding new terms used in this developer guide, and the definition or new meaning of each new term. For *fingerprint identification*, the term is the same, but the meaning is slightly different.

| Old Term | New Term | Definition |
|----------------------------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| biometric authentication | fingerprint authentication | See fingerprint verification. |
| fingerprint identification | | The process of comparing a supplied fingerprint credential with one or more enrollment fingerprint credentials based on a fingerprint provided by the user. If a matching stored fingerprint credential is found, the operation returns the user name. |
| fingerprint registration | fingerprint enrollment operation | An operation created for the purpose of enrolling a user's fingerprint. When the successful-operation message is received by the fingerprint-enabled application, a stored fingerprint credential is created, added to the user record, and stored for later comparison with a supplied fingerprint credential. The fingerprint enrollment operation can also be used to delete a stored fingerprint credential from a user record. |
| fingerprint verification | fingerprint authentication | The process of comparing a supplied fingerprint credential with a stored fingerprint credential based on the user name and a fingerprint provided by the user. If the two fingerprint credentials match, the operation returns the DigitalPersona Pro Secret or a message confirming that the fingerprint is from the user. |
| registration template | stored fingerprint credential | The output of a fingerprint enrollment operation, which is added to the user record and stored for later comparison with a supplied fingerprint credential. |
| verification template | supplied fingerprint credential | Data acquired by a fingerprint acquisition operation that is used for performing fingerprint authentication and fingerprint identification and for managing user data and secrets. |

Glossary

authentication

The process of verifying that a user, computer, service, or process is who or what it claims to be.

BAS

Stands for **Biometric Authentication Service**.

biometric authentication

See **fingerprint authentication**.

Local Biometric Authentication Service

The DigitalPersona Pro service that runs on the local machine.

comparison

The estimation, calculation, or measurement of similarity or dissimilarity between a supplied fingerprint credential and a stored fingerprint credential(s).

credential, credentials

Information used to gain access to a protected resource such as a Web site or a Windows program. Examples of credentials are a user name, a password, a fingerprint, and a smart card. Currently, the only credentials accepted by the DigitalPersona Pro and Pro SDK for user authentication are fingerprint credentials.

device events

Windows events generated by the fingerprint reader.

DigitalPersona Pro Secret

Application-specific user data that is stored in Microsoft Active Directory by the DigitalPersona Pro Server or locally by the local authentication server on the workstation. The secret is released to the calling application upon successful user authentication and can be used, for instance, to log on to programs and Web sites and for data encryption.

DPFPApi functions

API functions derived from the DPFPApi.h file.

DPSDApi functions

API functions derived from the DPSDAPI.h file.

enrollee

The user who is enrolling or has enrolled his or her fingerprint credentials in the system.

enrollment

See **fingerprint enrollment operation**.

event notifications

Notifications of Windows events generated by a fingerprint reader or by the DigitalPersona system and sent to a fingerprint-enabled application.

FEA

Stands for **fingerprint-enabled application**.

fingerprint

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

fingerprint acquisition operation

An operation created for the purpose of acquiring a supplied fingerprint credential to be used for fingerprint authentication, fingerprint identification, user management, and secret management.

fingerprint authentication

The process of comparing a supplied fingerprint credential with a stored fingerprint credential based on the user name and a fingerprint provided by the user. If the two fingerprint credentials match, the operation returns the DigitalPersona Pro Secret or a message confirming that the fingerprint is from the user.

fingerprint credential

Data returned after a successful scan of a finger by a fingerprint reader.

fingerprint-enabled application

An application developed using Pro API functions.

fingerprint enrollment operation

An operation created for the purpose of enrolling a user's fingerprint. When the successful-operation message is received by the fingerprint-enabled application, a stored fingerprint credential is created. The stored fingerprint credential can then be added to the user record and stored for later comparison with a supplied fingerprint credential. The fingerprint enrollment operation can also be used to delete a stored fingerprint credential from a user record.

fingerprint identification

The process of comparing a supplied fingerprint credential with one or more enrollment fingerprint credentials based on a fingerprint provided by the user. If a matching stored fingerprint credential is found, the operation returns the user name.

fingerprint image

A digital representation of a fingerprint.

fingerprint reader

A device that collects data from a user's fingerprint and converts it to a fingerprint credential.

fingerprint registration

See **fingerprint enrollment**.

fingerprint verification

See **fingerprint authentication**.

image

See **fingerprint image**.

match

The decision that the supplied fingerprint credential and the stored fingerprint credential being compared are from the same user.

non-match

The decision that the supplied fingerprint credential and the stored fingerprint credential being compared are not from the same user.

operation

A process created by the fingerprint-enabled application using Pro API functions for performing fingerprint enrollment or fingerprint acquisition.

operation events

Windows events generated by the DigitalPersona system and sent to a fingerprint-enabled application.

registration template

See **stored fingerprint credential**.

secret

See **DigitalPersona Pro Secret**.

secret content

The data contained in the DigitalPersona Pro Secret.

Server Biometric Authentication Service

The DigitalPersona Pro service that runs on the server.

stored fingerprint credential

The output of a fingerprint enrollment operation, which is added to the user record and stored for later comparison with a supplied fingerprint credential.

supplied fingerprint credential

Data acquired by a fingerprint acquisition operation that is used for performing fingerprint authentication and fingerprint identification and for managing user data and secrets.

user authentication

The process of verifying a user's identity based on one or more credentials provided by the user. Currently, the only credentials accepted by the DigitalPersona Pro and Pro SDK for user authentication are fingerprint credentials.

verification

See **fingerprint authentication.**

verification template

See **supplied fingerprint credential.**

Index

A

- Active Directory
 - role in typical fingerprint authentication operation 10
 - system requirement 4
- adding
 - a user to the kiosk identification list, function for 28
 - secret content, function for 52
 - stored fingerprint credential to user record, function for 30
- additional resources 3
 - online resources 3
 - related documentation 3
- Adobe Reader installer, location on product CD 6
- allocating necessary resources 46
- API
 - constants, definitions of 65
 - data structures
 - definitions of 61–64
 - See also* individual data structures by name
 - data types, definitions of 65
 - definitions files, location after installation 5
 - functions
 - definitions of 24–60
 - listed by category, name, page, and description 26
 - listed by name, page, and description 24
 - See also* individual functions by name
 - retrieving version information for 44
- API reference 24–65
- audience for this guide 1
- authentication, defined 67
- available hard-disk space requirement 3

B

- biometric authentication
 - See* fingerprint authentication

C

- CD-ROM requirement 3
- chapters, overview of 1
- checking
 - if user can be enrolled in the system using additional credentials, function for 48
 - if user can be enrolled in the system, function for 47
- comparative glossary 66
- comparison, defined 67
- concepts used in Pro SDK 4.2 7
- constants, definitions of 65

- conventions, document
 - See* document conventions
- Courier bold typeface, use of 2
- creating
 - a fingerprint acquisition operation, function for 31
 - a fingerprint enrollment operation, function for 35
 - stored fingerprint credential 12
 - typical workflow illustrated 12
 - typical workflow tasks 13
- credential, defined 67

D

- data flow
 - for Pro SDK 10
 - for typical fingerprint authentication operation
 - explained 10
 - illustrated 10
- data structures
 - definitions of 61–64
 - See also* individual data structures by name
- data types, definitions of 65
- definitions files
 - functions, data structures, types, and constants extracted from 24–65
 - location after installation 5
- deleting
 - secret content, function for 39
 - stored fingerprint credential
 - from user record, function for 30
 - function for 38
 - typical workflow 14
 - illustrated 14
 - tasks 15
- destroying
 - fingerprint acquisition operation, function for 40
 - fingerprint enrollment operation, function for 40
- developer guide
 - location after installation 5
 - location on product CD 6
 - overview of chapters 1
- device events
 - defined 67
 - values for 32, 36
- devices
 - enumerating, function for 41
 - retrieving information about, function for 42

- dialog box, displaying
 - for fingerprint authentication 9
 - function for 58
 - for fingerprint authentication for logon user 9
 - function for 57
 - for fingerprint identification 9
 - function for 55
 - for fingerprint identification with authentication for logon user 9
 - function for 54
- DigitalPersona Developer Connection Forum, URL for 3
- DigitalPersona Pro for Active Directory 4.2
 - administrator guide 3
 - supported product 4
 - terminology in administrator guide compared with Pro SDK 4.2 7, 66
- DigitalPersona Pro SDK
 - See SDK
- DigitalPersona Pro Secret
 - defined 67
 - deleting content from, function for 39
 - returning, function for 28
 - writing content to, function for 52
- DigitalPersona Pro Server, in Pro SDK data flow 10
- DigitalPersona Pro Workstation 4.2
 - system requirement 4
 - See also DigitalPersona Pro Workstation
- DigitalPersona Pro Workstation, in Pro SDK data flow 10
- DigitalPersona U.are.U fingerprint reader
 - supported product 4
 - system requirement 3
- disc drive requirement 3
- document conventions 2
 - naming 2
 - notational 2
 - typographical 2
- documentation, related 3
- DP_AUTHENTICATION_INFO data structure, defined 61
- DP_DEVICE_INFO data structure
 - defined 62
 - information returned in by DPFPGetDeviceInfo 42
- DP_DEVICE_VERSION data structure, defined 63
- DP_HW_INFO data structure, defined 63
- DP_MAX_USB_STRING_SIZE constant, defined 65
- DP_PRODUCT_VERSION data structure
 - defined 64
 - information returned in by DPFPGetVersion 44
- DP_USER_INFO_0 data structure
 - defined 64
 - information returned in by DPFPGetUserInfo 42
- DP_USER_NAME_SAM_COMPATIBLE format
 - defined 43
 - names returned in by DPFPIdentify function 45
 - names returned in by DPSDIdentifyLogonUser function 55
 - names returned in by DPSDIdentifyUser function 56
- DP_USER_NAME_TYPE type parameter, values for 43
- DPFPAddUserToIdentList function, defined 28
- DPFPApi functions
 - defined 67
 - freeing memory allocated by 30
- DPFPAuthenticate function
 - defined 28
 - important notice to free allocated memory 28
 - using in typical fingerprint authentication workflow
 - with UI support 19
 - without UI support 17
 - using in typical fingerprint identification with authentication workflow
 - with UI support 23
 - without UI support 21
- DPFPBufferFree function
 - defined 30
 - using in typical fingerprint authentication workflow
 - with UI support 19
 - without UI support 17
 - using in typical fingerprint enrollment operation for creating stored fingerprint credential 13
 - using in typical fingerprint identification with authentication workflow
 - with UI support 23
 - without UI support 21
- DPFPCommitRegistrationChanges function
 - defined 30
 - using in typical fingerprint enrollment operation for creating stored fingerprint credential 13
 - for deleting stored fingerprint credential 15
- DPFPCreateAcquisition function
 - defined 31
 - important notice to free allocated memory 31
 - using in typical fingerprint authentication workflow
 - without UI support 17
 - using in typical fingerprint identification with authentication workflow without UI support 21
 - values of wParam and lParam for event notifications 32
- DPFPCreateRegistration function
 - defined 35
 - important notice to free allocated memory 35
 - using in typical fingerprint enrollment operation

- for creating stored fingerprint credential 13
 - for deleting stored credential 15
- values of wParam and lParam for event notifications 36
- DPFPDeleteFinger function
 - defined 38
 - important notice to complete operation by calling DPFPCommitRegistrationChanges 38
 - using in typical fingerprint enrollment operation for deleting stored credential 15
- DPFPDeleteSecret function, defined 39
- DPFPDestroyAcquisition function
 - defined 40
 - using in typical fingerprint authentication workflow without UI support 17
 - using in typical fingerprint identification with authentication workflow without UI support 22
- DPFPDestroyRegistration function
 - defined 40
 - using in typical fingerprint enrollment operation for creating stored fingerprint credential 13
 - for deleting stored credential 15
- DPFPEnumerateDevices function
 - defined 41
 - important notice to free allocated memory 41
 - using in typical fingerprint authentication workflow without UI support 17
 - using in typical fingerprint enrollment operation for creating stored fingerprint credential 13
 - using in typical fingerprint identification with authentication workflow without UI support 21
- DPFPGetDeviceInfo function, defined 42
- DPFPGetUserInfo function
 - defined 42
 - important notice to free allocated memory 42
- DPFPGetVersion function, defined 44
- DPFPIdentify function
 - defined 44
 - important notice to free allocated memory 44
 - using in typical fingerprint identification with authentication workflow without UI support 21
- DPFPIdentifyUser function
 - defined 45
 - important notice to free allocated memory 45
- DPFPInit function
 - defined 46
 - important notice to call before any other functions except DPFPBufferFree 46
 - important notice to pair with DPFPTerm 46
- using in typical fingerprint authentication workflow
 - with UI support 19
 - without UI support 17
- using in typical fingerprint enrollment operation
 - for creating stored fingerprint credential 13
 - for deleting stored fingerprint credential 15
- using in typical fingerprint identification with authentication workflow
 - with UI support 23
 - without UI support 21
- DPFPIsRegistrationAllowed, defined 47
- DPFPIsRegistrationAllowedEx function, defined 48
- DPFPStartAcquisition function
 - defined 49
 - important notice to pair with DPFPStopAcquisition 49
 - using in typical fingerprint authentication workflow without UI support 17
 - using in typical fingerprint identification with authentication workflow without UI support 21
- DPFPStartRegistration function
 - defined 49
 - important notice to pair with DPFPStopRegistration 50
 - using in typical fingerprint enrollment operation for creating stored credential 13
- DPFPStopAcquisition function
 - defined 51
 - using in typical fingerprint authentication workflow without UI support 17
 - using in typical fingerprint identification with authentication workflow without UI support 21
- DPFPStopRegistration function
 - defined 51
 - using in typical fingerprint enrollment operation for creating stored fingerprint credential 13
- DPFPTerm function
 - defined 52
 - using in typical fingerprint authentication workflow
 - with UI support 19
 - without UI support 18
 - using in typical fingerprint enrollment operation
 - for creating stored fingerprint credential 13
 - for deleting stored credential 15
 - using in typical fingerprint identification with authentication workflow
 - with UI support 23
 - without UI support 22
- DPFPWriteSecretContent function, defined 52

DPSDApi functions
 defined 67
 freeing memory allocated by 53

DPSDBufferFree function
 defined 53
 using in typical fingerprint identification with authentication workflow with UI support 23

DPSDIdentifyLogonUser function 9
 defined 54
 important notice to free allocated memory 54

DPSDIdentifyUser function 9
 defined 55
 important notice to free allocated memory 55
 using in typical fingerprint identification with authentication workflow with UI support 23

DPSDVerifyLogonUser function 9
 defined 57

DPSDVerifyUser function 9
 defined 58
 using in typical fingerprint authentication workflow with UI support 19

E

End User License Agreement, location on product CD 6
 enrollee, defined 67
 enrollment
 See fingerprint enrollment operation

enumerating available fingerprint readers
 function for 41

EULA
 See End User License Agreement

event notifications 49
 defined 67
 subscribed to by
 DPFStartAcquisition 32
 DPFStartRegistration 36
 subscribing fingerprint-enabled application to for fingerprint acquisition operations, function for 49
 unsubscribed from by
 DPFStopAcquisition 32
 DPFStopRegistration 36
 unsubscribing fingerprint-enabled application from receiving
 for fingerprint acquisition operations, function for 51
 for fingerprint enrollment operations, function for 51

F

FEA
 See fingerprint-enabled application

files and folders
 installed for SDK 5
 location on product CD 6

fingerprint acquisition operation 8
 creating, function for 31
 defined 67
 destroying, function for 40
 freeing resources associate with, function for 40
 priorities 8
 priority values, defined 32
 workflows 15–23

fingerprint authentication 8
 defined 67
 performing
 function for 28
 on local machine, function for 45
 with UI support 9
 function for 58
 typical workflow illustrated 18
 typical workflow tasks 19
 with UI support for logon user 9
 function for 57
 without UI support
 typical workflow illustrated 16
 typical workflow tasks 17

fingerprint credential
 defined 67
 See also stored fingerprint credential, supplied fingerprint credential

fingerprint enrollment operation 7
 checking if user can be enrolled, additional credentials required, function for 48
 checking if user can be enrolled, function for 47
 creating, function for 35
 defined 68
 destroying, function for 40
 freeing resources associate with, function for 40
 important notice to complete by calling DPFCommitRegistrationChanges 50
 priority 8, 49
 saving changes made during, function for 30
 subscribing fingerprint-enabled application to receive event notifications, function for 49
 unsubscribing fingerprint-enabled application to receive event notifications, function for 51
 workflows 11–15

- fingerprint enrollment operation workflow
 - creating stored fingerprint credential 12
 - illustrated 12
 - tasks 13
 - deleting stored fingerprint credential 14
 - illustrated 14
 - tasks 15
- fingerprint identification 8
 - defined 68
 - performing
 - function for 44
 - with fingerprint authentication with UI support
 - function for 55
 - typical workflow illustrated 22
 - typical workflow tasks 23
 - with fingerprint authentication with UI support for
 - login user 9
 - function for 54
 - with fingerprint authentication without UI support
 - typical workflow illustrated 20
 - typical workflow tasks 21
- fingerprint image
 - defined 68
 - quality values, defined 33, 36
- fingerprint reader
 - defined 68
 - enumerating available, function for 41
 - retrieving information about, function for 42
 - supported product 4
 - system requirement 3
- fingerprint registration
 - See fingerprint enrollment operation
- fingerprint verification
 - See fingerprint authentication
- fingerprint, defined 67
- fingerprint-enabled application 7
 - defined 68
 - fingerprint acquisition operation event notifications
 - subscribing to, function for 49
 - unsubscribing from, function for 51
 - fingerprint enrollment operation event notifications
 - subscribing to, function for 49
 - unsubscribing from, function for 51
 - role in typical Pro SDK fingerprint authentication
 - operation 10
- folders and files
 - installed for SDK 5
 - location on product CD 6
- freeing
 - memory allocated by
 - DPFPApi functions 30
 - DPSDApi functions 53
 - resources allocated by
 - DPFPInit, function for 52
 - resources associated with
 - fingerprint acquisition operation, function for 40
 - fingerprint enrollment operation, function for 40
- functions
 - definitions of 24–60
 - listed by category name, page, and description 26
 - listed by name, page, and description 24
 - See also individual functions by name
- G**
 - getting information about
 - a fingerprint reader, function for 42
 - a user, function for 42
 - the Pro API version, function for 44
 - glossary
 - comparative 66
 - main 67–69
- H**
 - hard disk
 - available space requirement 3
 - SDK files and folders installed on 5
 - hardware
 - DigitalPersona products supported 4
 - requirements 3
 - HDPCREDENTIAL data type, defined 65
 - HDPOPERATION data type, defined 65
 - header files
 - definitions extracted from 24–65
 - location after installation 5
- I**
 - image
 - See fingerprint image
 - import library files, location after installation 5
 - important notation, defined 2
 - important notice
 - all fingerprint enrollment operations must be
 - completed by calling
 - DPFPCommitRegistrationChanges 30, 50
 - developer must provide functionality for acquiring
 - user name 10, 17
 - DPFPInit must be called before calling any other
 - functions except DFPFBufferFree 46
 - DPFPInit must be paired with DFPFTerm 46

DPFPStartAcquisition must be paired with
 DPFPStopAcquisition 49
 DPFPStartRegistration must be paired with
 DPFPStopRegistration 50
 memory allocated by DPFPAuthenticate must be
 freed by calling DPFPBufferFree 28
 memory allocated by DPFPCreateAcquisition must be
 freed by calling DPFPDestroyAcquisition 31
 memory allocated by DPFPCreateRegistration must
 be freed by calling DPFPDestroyRegistration 35
 memory allocated by DPFPEnumerateDevices must
 be freed by calling DPFPBufferFree 41
 memory allocated by DPFPGetUserInfo must be freed
 by calling DPFPBufferFree 42
 memory allocated by DPFPIdentify must be freed by
 calling DPFPBufferFree 44
 memory allocated by DPFPIdentifyUser must be freed
 by calling DPFPBufferFree 45
 memory allocated by DPSDIdentifyLogonUser must
 be freed by calling DPSDBufferFree 54
 memory allocated by DPSDIdentifyUser must be freed
 by calling DPSDBufferFree 55
 initializing necessary resources 46
 installation of product 5
 installing the SDK 5
 Internet Explorer, system requirement 4
 italics typeface, uses of 2

K

kiosk

- adding a user to the list, function for 28
- checking if user can be enrolled, function for 48

kiosk identification list, adding a user to, function for 28

L

library files, location after installation 5

Local Biometric Authentication Service

- defined 67
- in operations 7
- operation-processing hierarchy 8
- role in typical Pro SDK fingerprint authentication
 operation 10

logon token, returning, function for 54, 57

logon user

- displaying dialog box for fingerprint authentication 9
- displaying dialog box for fingerprint identification
 with authentication 9

IParam values for event notifications subscribed to by

- DPFPStartAcquisition 32
- DPFPStartRegistration 36

M

managing

- fingerprint readers (devices) 41
- secrets 8
- user data 8

match, defined 68

memory

- allocated by DPFPApi functions, freeing, function
 for 30
- allocated by DPSDApi functions, freeing, function
 for 53

memory requirement 3

Microsoft Active Directory

- See Active Directory

Microsoft Internet Explorer

- See Internet Explorer

Microsoft Windows

- See Windows

N

name types, user, values for 43

naming conventions 2

non-match, defined 68

notational conventions 2

note notation, defined 2

O

One Touch Internet 4

OneTouch SignOn 4

online resources 3

operating-system requirement 4

operation 7

- defined 68
- fingerprint acquisition 8
- fingerprint enrollment 7
- workflows 11–23

operation events, defined 68

overview

- of chapters 1
- of SDK 7

P

performing

- fingerprint authentication 8
- for the logon user with UI support 9
- function for 57
- function for 28
- on local machine, function for 45

- with UI support 18
 - function for 58
 - typical workflow illustrated 18
 - typical workflow tasks 19
- without UI support 16
 - typical workflow illustrated 16
 - typical workflow tasks 17
- fingerprint identification 8
 - function for 44
 - with UI support, function for 55
- fingerprint identification with authentication
 - for the logon user with UI support 9
 - function for 54
 - with UI support 22
 - typical workflow illustrated 22
 - typical workflow tasks 23
 - without UI support 20
 - typical workflow illustrated 20
 - typical workflow tasks 21
- priorities 8
 - values defined 32
- Pro Kiosk
 - See kiosk
- Pro SDK
 - See SDK
- processor requirement 3
- product CD, files and folders on 6
- product installation 5

Q

- quality values for fingerprint images 33, 36

R

- RAM requirement 3
- registration template
 - See stored fingerprint credential
- requirements, system
 - See system requirements
- resources
 - allocating and initializing necessary 46
 - associated with fingerprint acquisition operation,
 - freeing, function for 40
 - associated with fingerprint enrollment operation,
 - freeing, function for 40
- resources, additional
 - See additional resources
- resources, online
 - See online resources

- retrieving information about
 - a fingerprint reader, function for 42
 - a user, function for 42
 - the Pro API version, function for 44

S

- sample applications and projects, location after installation 5
- saving changes made during a fingerprint enrollment operation, function for 30
- SDK
 - concepts and terminology 7
 - data flow 10
 - illustrated 10
 - files and folders installed on hard disk 5
 - files and folders on product CD 6
 - installing 5
- secret
 - See DigitalPersona Pro Secret
- secret content
 - adding, function for 52
 - defined 68
 - deleting, function for 39
 - writing, function for 52
- secrets, managing 8
- Server Biometric Authentication Service
 - defined 68
 - role in typical Pro SDK fingerprint authentication operation 10
- software
 - DigitalPersona products supported 4
 - requirements 3
- stored fingerprint credential 7
 - adding to user record, function for 30
 - defined 68
 - deleting from user record, function for 30
 - deleting, function for 38
 - workflow for creating 12
 - illustrated 12
 - tasks 13
 - workflow for deleting 14
 - illustrated 14
 - tasks 15
- subscribing
 - fingerprint-enabled application to receive fingerprint acquisition operation event notifications,
 - function for 49
 - fingerprint-enabled application to receive fingerprint enrollment operation event notifications,
 - function for 49

subscribing fingerprint-enabled application to
 for fingerprint enrollment operations 49

supplied fingerprint credential
 creating
 in typical fingerprint authentication workflow
 without UI support 17
 in typical fingerprint identification with fingerprint
 authentication workflow without UI
 support 21
 in typical Pro SDK fingerprint authentication
 operation 10
 defined 68
 purpose of 7, 8
 supported DigitalPersona products 4
 system requirements 3

T

target audience for this guide 1

terminating resources allocated by DPFPTerm 52

terminology used in Pro SDK 4.2 7
 compared with Pro for Active Directory 4.2
 Administrator Guide 7, 66

typefaces, uses of
 Courier bold 2
 italics 2

typographical conventions 2

U

UI support 9
 for fingerprint authentication 9
 function for 58
 for fingerprint authentication for logon user 9
 function for 57
 for fingerprint identification 9
 function for 55
 for fingerprint identification with authentication for
 logon user 9
 function for 54
 performing fingerprint authentication 18
 typical workflow illustrated 18
 typical workflow tasks 19
 performing fingerprint identification with
 authentication 22
 typical workflow illustrated 22
 typical workflow tasks 23

uMsg parameter
 possible values for DPFPCreateAcquisition 32
 possible values for DPFPCreateRegistration 36

unsubscribing
 fingerprint-enabled application from receiving
 fingerprint acquisition operation event
 notifications, function for 51
 fingerprint-enabled application from receiving
 fingerprint enrollment operation event
 notifications, function for 51

updates for DigitalPersona software products, URL for
 downloading 3

URL
 DigitalPersona Developer Connection Forum 3
 Updates for DigitalPersona Software Products 3

USB port, system requirement 3

user
 checking if can be enrolled in the system using
 additional credentials, function for 48
 checking if can be enrolled in the system, function
 for 47
 retrieving information about, function for 42

user authentication, defined 68

user data, managing 8

user name types, values for 43

user record
 adding stored fingerprint credential to, function for 30
 deleting secret content from, function for 39
 deleting stored fingerprint credential from, function
 for 30
 writing secret content to, function for 52

usri0_fp_credential_mask data field, values for 64

V

verification
 See fingerprint authentication

verification template
 See supplied fingerprint credential

W

warning notation, defined 2

Web site
 DigitalPersona Developer Connection Forum 3
 Updates for DigitalPersona Software Products 3

Windows installer package, location on product CD 6

Windows NT 4.0 account name, value for 43

WN_COMPLETED message
 defined 32
 sent during typical fingerprint authentication
 workflow without UI support 16
 sent during typical fingerprint identification with
 authentication workflow without UI support 20

Index

- WN_REGISTRATION_COMPLETED message
 - defined 36
 - receiving during typical fingerprint enrollment operation for creating stored fingerprint credential 13
- workflow
 - for creating a stored fingerprint credential
 - illustrated 12
 - tasks 13
 - for creating stored fingerprint credential 12
 - for deleting a stored fingerprint credential 14
 - illustrated 14
 - tasks 15
 - for performing fingerprint authentication
 - with UI support 18
 - illustrated 18
 - tasks 19
 - without UI support 16
 - illustrated 16
 - tasks 17
 - for performing fingerprint identification with authentication
 - with UI support 22
 - illustrated 22
 - tasks 23
 - without UI support 20
 - illustrated 20
 - tasks 21
- wParam values for event notifications subscribed to by
 - DPFPStartAcquisition 32
 - DPFPStartRegistration 36
- writing secret content, function for 52