

DigitalPersona, Inc.

One Touch[®] for Linux SDK

Version 1.1

Developer Guide



digitalPersona.

DigitalPersona, Inc.

© 1996–2008 DigitalPersona, Inc. All Rights Reserved.

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

DigitalPersona, One Touch, and U.are.U are trademarks of DigitalPersona, Inc., registered in the United States and other countries. Adobe and Adobe Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. All other trademarks are the property of their respective owners.

This DigitalPersona One Touch for Linux 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

Technical support is available through the DigitalPersona Developer Connection at www.digitalpersona.com/webforums, where you can search for answers to questions posted by other developers and post your own questions. You can also purchase a Developer Support package at our web store, <http://buy.digitalpersona.com>.

Feedback

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

TechPubs@digitalpersona.com

or

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

Table of Contents

1	Introduction	1
	Target Audience	1
	Chapter Overview	1
	Document Conventions	2
	Notational Conventions	2
	Naming Conventions	2
	Typographical Conventions	3
	Additional Resources	3
	Related Documentation	3
	Online Resources	4
	Supported DigitalPersona Products	4
2	Quick Start	5
	Install the Software	5
	Using the Sample Application	5
3	One Touch for Linux SDK Overview	9
	One Touch for Linux SDK Terminology and Concepts	9
	Biometric System	9
	Fingerprint	9
	Fingerprint Recognition	10
	Fingerprint Enrollment	10
	Fingerprint Verification	10
	False Positives and False Negatives	11
	Components of the SDK	12
	Data Flow	12
	Device Component	14
	Initialization	15
	Operation	15
	Clean-up	15
	Fingerprint Recognition Component	16
	Fingerprint Enrollment	16
	Fingerprint Verification	23

4	One Touch for Linux API Reference	29
	Functions	29
	DPFPBufferFree	31
	DPFPEnumerateDevices	31
	DPFPGetDeviceInfo	32
	DPFPGetEvent	33
	DPFPInit	35
	DPFPSubscribe	35
	DPFPTerm	36
	DPFPUnsubscribe	37
	FX_closeContext	38
	FX_createContext	38
	FX_extractFeatures	39
	FX_getDisplayImage	42
	FX_getFeaturesLen	43
	FX_getVersionInfo	44
	FX_init	45
	FX_terminate	46
	MC_closeContext	46
	MC_createContext	47
	MC_generateRegFeatures	47
	MC_getFeaturesLen	49
	MC_getSecurityLevel	50
	MC_getSettings	50
	MC_getVersionInfo	51
	MC_init	51
	MC_setSecurityLevel	52
	MC_terminate	53
	MC_verifyFeaturesEx	53
	Data Structures	56
	dp_device_event_t	56
	dp_device_info_t	57
	dp_hw_info_t	58
	dp_version_t	58
	MC_SETTINGS	59
	FT_VERSION_INFO	59
5	One Touch for Linux API Return Codes	60
	Device Component	60
	Fingerprint Recognition Component	60

6	Redistribution	62
	Device Component Libraries	62
	Fingerprint Recognition Component Libraries	62
	User Mode Driver Libraries	63
	Kernel Mode Driver Source Files	64
	Kernel Mode Driver Files	65
	Fingerprint Reader Documentation	66
	Hardware Warnings and Regulatory Information	66
	Fingerprint Reader Use and Maintenance Guide	66
	Glossary	67
	Index	70

User recognition based on fingerprints is one of the best ways of incorporating security and convenience into computer systems. Fingerprint recognition has been added to hundreds of applications in the fields of finance, healthcare, and government. For example, fingerprint recognition is used in these areas to control physical access or to record time and attendance. These applications have saved time and money in a variety of ways, such as by eliminating the need for password replacement and gatekeepers to secure facilities and by ensuring compliance to banking and healthcare policies.

The One Touch® for Linux SDK continues in the tradition of DigitalPersona SDKs by providing versatile, easy-to-use, and best-of-breed technology. The SDK provides developers with a simple way to obtain fingerprint images using a DigitalPersona fingerprint reader, to extract the distinctive characteristics from these images, and to use them to enroll a person's fingerprint(s) for later comparison with other enrolled fingerprints with the highest level of accuracy available in today's market.

The One Touch for Linux SDK contains documentation and samples that enable Linux developers to create a wide variety of custom applications that incorporate the power of the DigitalPersona Fingerprint Recognition Engine for use with the award-winning U.are.U® Fingerprint Reader.

Target Audience

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

Chapter Overview

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

Chapter 2, *Quick Start*, provides a quick introduction to the One Touch for Linux SDK using the sample application provided as part of the SDK.

Chapter 3, *One Touch for Linux SDK Overview*, introduces One Touch for Linux SDK terminology and concepts, shows how data flows among the various One Touch for Linux SDK components, and includes typical workflow diagrams and explanations of the One Touch for Linux API functions used to perform the operations in the workflows.

Chapter 4, *One Touch for Linux API Reference*, defines the functions and data structures that are part of the One Touch for Linux API.

Chapter 5, *One Touch for Linux API Return Codes*, defines the codes returned by the One Touch for Linux API functions.

Chapter 6, *Redistribution*, identifies the files that you may redistribute according to the End User License Agreement (EULA) provided in the One Touch for Linux SDK product package.

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

Document Conventions

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

Notational Conventions

The following notational conventions are used throughout this guide:

NOTE: Notes provide reminders, tips, or suggestions that supplement the material included in this guide.

IMPORTANT: Important notations contain significant information about system behavior, 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.

Naming Conventions

The *DPFP* prefix used in device component API functions, type definitions, and constants stands for *DigitalPersona Fingerprint*.

The *FX* prefix used in fingerprint feature extraction module API functions, type definitions, and constants stands for *Feature Extraction*.

The *MC* prefix used in fingerprint comparison module API functions, type definitions, and constants stands for *Matching*.

Typographical Conventions

The following typographical conventions are used in this guide:

Typeface	Purpose	Example
Courier bold	Used to indicate computer programming code	The uTimeout parameter is in milliseconds. Initialize the device component library 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.	This section includes illustrations of <i>typical</i> fingerprint enrollment and verification workflows. (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)
Bold	Used for text that you enter from the keyboard or for keystrokes	At the prompt, run depmod . Press E to exit the sample application.

Additional Resources

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

Related Documentation

Subject	Document
Fingerprint recognition, including the history and basics of fingerprint recognition and the advantages of DigitalPersona's Fingerprint Recognition Engine	The DigitalPersona White Paper: Guide to Fingerprint Recognition (Fingerprint_Guide.pdf located in the docs directory in the One Touch for Linux SDK product package)
Late-breaking news about the product	The Readme.txt files provided in the root directory of the product package as well as in some subdirectories

Online Resources

Web Site Description	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

Supported DigitalPersona Products

The One Touch for Linux SDK supports the following DigitalPersona product:

- DigitalPersona U.are.U 4000B Fingerprint Reader, Revisions 100 and 101

This chapter provides a quick introduction to the One Touch for Linux SDK using the sample application provided as part of the One Touch for Linux SDK.

IMPORTANT: You must be root to install the SDK and to run the sample application.

Install the Software

Before you can use the sample application, you must install the One Touch for Linux SDK. Refer to the installation guide for distribution/kernel version that you are using.

Using the Sample Application

Sample project files are installed in the `/opt/DigitalPersona/OneTouchSDK/sample` directory. A simple, non-persistent database and temporary storage system are built in to the project for demonstration purposes only. You must provide these functionalities in your application, as they are not part of the One Touch for Linux SDK.

The sample application that is compiled from the project files uses the One Touch for Linux API functions defined in the `dpFtrEx.h`, `dpMatch.h`, `dpDefs.h`, and `dpfp_api.h` files installed in the `/opt/DigitalPersona/OneTouchSDK/include` directory.

By performing the exercises in this section, you will

- Add yourself (and others) to the built-in database
- Perform fingerprint enrollment (*page 16*)
- Perform fingerprint verification (*page 23*)
- Identify the fingerprint reader(s) connected to your computer (*page 15*)
- Select a fingerprint reader(s) (*page 15*)
- Exit the sample application

To compile and run the sample application

1. Change to the `/opt/DigitalPersona/SDK/sample` directory.
2. Run **make**.

3. Run **make run**.

The following menu appears:

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

To add yourself (and others) to the database

1. Press **1**.
2. At the **Enter a name** prompt, enter your name.

Your name is stored in the built-in database by the sample application.

You can add other people to the database by repeating steps 1 and 2 and using their names instead of yours.

To perform fingerprint enrollment

1. Press **2**.
2. At the **Enter the name of the person to be enrolled** prompt, enter your name.
3. Choose the finger you want to enroll, for example, press **7** to enroll your right index finger.

NOTE: To complete the fingerprint verification exercise on *page 7*, you should not enroll at least one of your fingers.

4. At the **Touch the reader with the appropriate finger, or press C to cancel the operation and return to the previous menu** prompt, touch the reader with your right index finger, as shown in the figure on the right.



A fingerprint feature set is created and then stored in volatile memory by the application.

5. Repeat step 4 until four consecutive fingerprint samples are acquired and the following message appears:

The finger was enrolled.

A fingerprint template is created for your finger and then stored in the built-in database by the application.

You can enroll additional fingers for yourself by repeating steps 1 through 5, or you can enroll other people that you added to the database using their names and fingers instead of yours.

To perform fingerprint verification

1. Press **3**.
2. At the **Enter the name of the person to be verified** prompt, enter the name of a person who is not in the database.

The following message appears:

```
<Name> was not found in the database.
```

3. Press **3**.
4. At the **Enter the name of the person to be verified** prompt, enter your name.
5. At the **Touch the reader with the appropriate finger, or press C to cancel the operation and return to the previous menu** prompt, touch the reader with any finger that you did not enroll in the previous exercise.

A fingerprint comparison module function returns a decision of non-match (*page 53*), and the following message appears:

```
No matching finger for <Your Name> was found.
```

6. Press **3**.
7. At the **Enter the name of the person to be verified** prompt, enter your name.
8. At the **Touch the reader with the appropriate finger, or press C to cancel the operation and return to the previous menu** prompt, touch the reader with your right index finger.

The same fingerprint comparison module function returns a decision of match, and a message similar to the following appears. The message includes the identity of the matching finger and the achieved false accept rate (FAR) (*page 11*).

```
Matching finger: Right index finger. Achieved FAR: 3.687697e-41.  
Matching finger for <Your Name> was found.
```

For any fingers that you have not enrolled, you can repeat steps 3 through 5 to return a non-match decision. For any fingers that you have enrolled, you can repeat steps 6 through 8 to return a match decision. In addition, you can perform verification for other people using their names and fingers instead of yours.

To display a list of all available readers

- Press **4**.

To choose a fingerprint reader(s)

1. Press **5**.
2. Press the number next to a specific fingerprint reader to choose it, or press the number next to **Any available readers** to choose all (other) fingerprint readers connected to your computer.

To exit the sample application

- Press **E**.

This chapter introduces One Touch for Linux SDK terminology and concepts, shows how data flows among the various One Touch for Linux SDK components, and includes typical workflow diagrams and explanations of the One Touch for Linux API functions used to perform the tasks in the workflows.

One Touch for Linux SDK Terminology and Concepts

This section defines the terminology and concepts used throughout this guide. Only a brief discussion of fingerprint recognition is included. For more details on this subject, refer to the “DigitalPersona White Paper: Guide to Fingerprint Recognition” included in the One Touch for Linux SDK product package.

Biometric System

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

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

Biometric characteristics are more secure than *token-based systems*, which recognize a person according to something that the person “has,” such as a card or a key, or *knowledge-based systems*, which recognize a person according to what the person “knows,” such as a password or an account number.

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

Fingerprint

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

Fingerprint Recognition

The DigitalPersona fingerprint recognition system uses the processes of fingerprint enrollment and fingerprint verification, which are illustrated in the block diagram in Figure 1 on *page 11*. Some of the tasks in these processes are done by the *fingerprint reader* and its driver; some are accomplished using One Touch for Linux API functions, which use the Engine; and some are provided by your software application and/or hardware.

Fingerprint Enrollment

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

1. *Obtain the enrollee's identifier (*Subject Identifier*).
2. *Capture the enrollee's fingerprint using the fingerprint reader.
3. Extract the *fingerprint feature set* for the purpose of enrollment from the fingerprint sample.
4. Repeat steps 2 and 3 until you have enough fingerprint feature sets to create a fingerprint template.
5. Create a fingerprint template.
6. *Associate the fingerprint template with the enrollee through a Subject Identifier, such as a user name, email address, or employee number.
7. *Store the fingerprint template, along with the Subject Identifier, for later comparison.

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

Fingerprint Verification

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

1. *Obtain the Subject Identifier of the person to be verified.
2. *Capture a fingerprint sample using the fingerprint reader.
3. Extract a fingerprint feature set for the purpose of verification from the fingerprint sample.
4. *Retrieve the fingerprint template associated with the Subject Identifier from your repository.
5. Perform a *one-to-one comparison* between the fingerprint feature set and the fingerprint template, and make a decision of *match* or *non-match*.

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

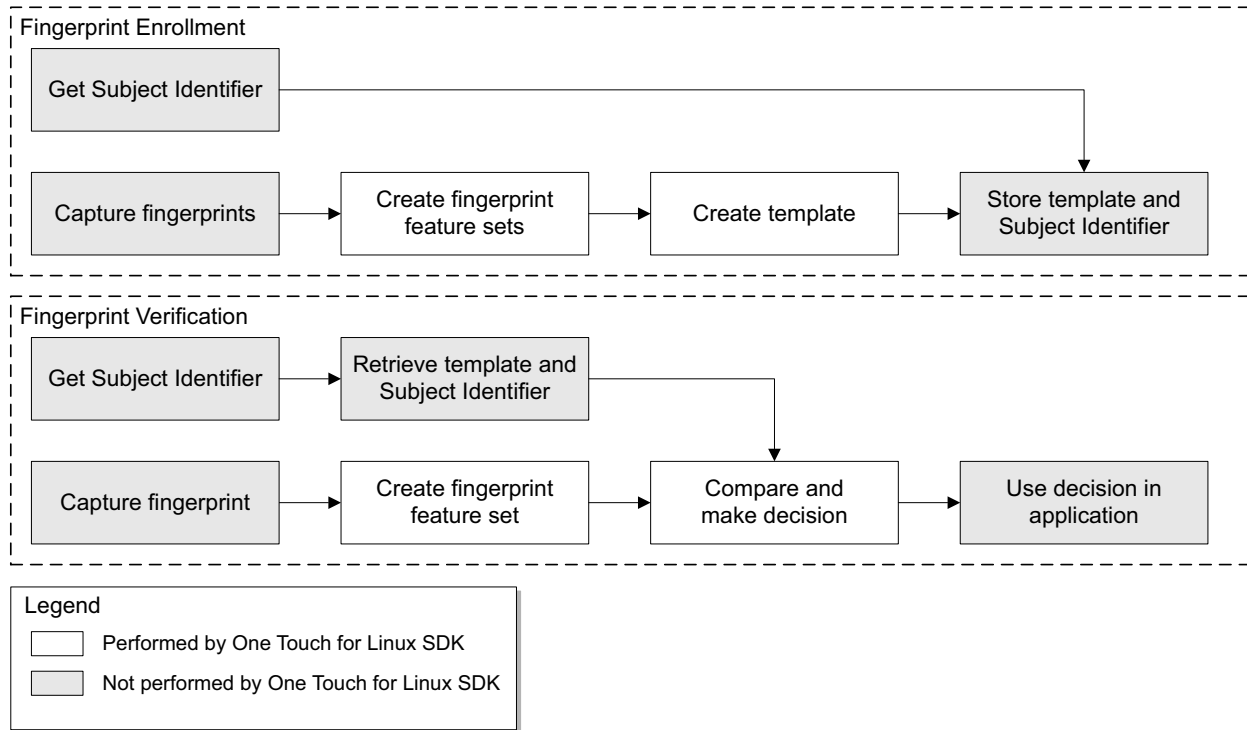


Figure 1. DigitalPersona fingerprint recognition system

False Positives and False Negatives

Although fingerprint recognition systems provide many security and convenience advantages over traditional methods of recognition, they are not perfect. During verification, occasionally a person who is legitimately enrolled is rejected by the system (a false negative decision), and sometimes a person who is not enrolled is accepted by the system (a false positive decision).

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

A One Touch for Linux API function enables you to set the value of the FAR, or the security level, to accommodate the needs of your application. In some applications, such as an access system to a confidential site or database, a lower FAR is required. In other applications, such as an entry system to an entertainment theme park, security may not be as significant as accessibility, and it may be preferable to decrease the FRR at the expense of an increased FAR.

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

Components of the SDK

The One Touch for Linux SDK consists of the following two components:

- Device component

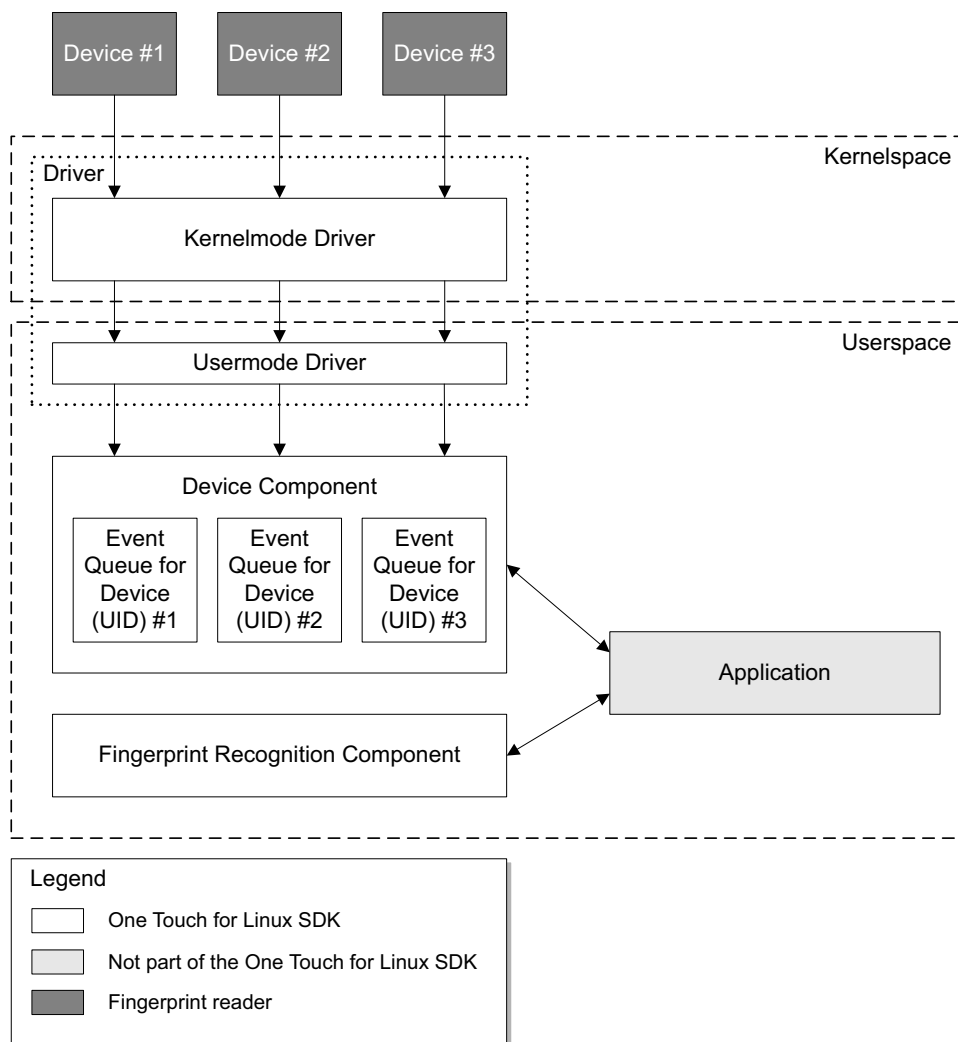
The device component directs fingerprint reader (device) data and events to your application.

- Fingerprint recognition component

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

Data Flow

Figure 2 illustrates how data flows among the fingerprint reader(s), the two components of the One Touch for Linux SDK, and the application that you develop using One Touch for Linux API functions. The One Touch for Linux API libraries perform fingerprint enrollment and verification. These tasks are implemented via the API functions defined in the `dpFtrEx.h`, `dpMatch.h`, `dpDefs.h`, and `dpfp_api.h` files located in the `/opt/DigitalPersona/OneTouchSDK/include` directory.

**Figure 2.** One Touch for Linux SDK data flow

Device Component

The device component workflow is represented in *Figure 3* and is followed by explanations of the One Touch for Linux API functions that are used to perform the tasks in the workflow.

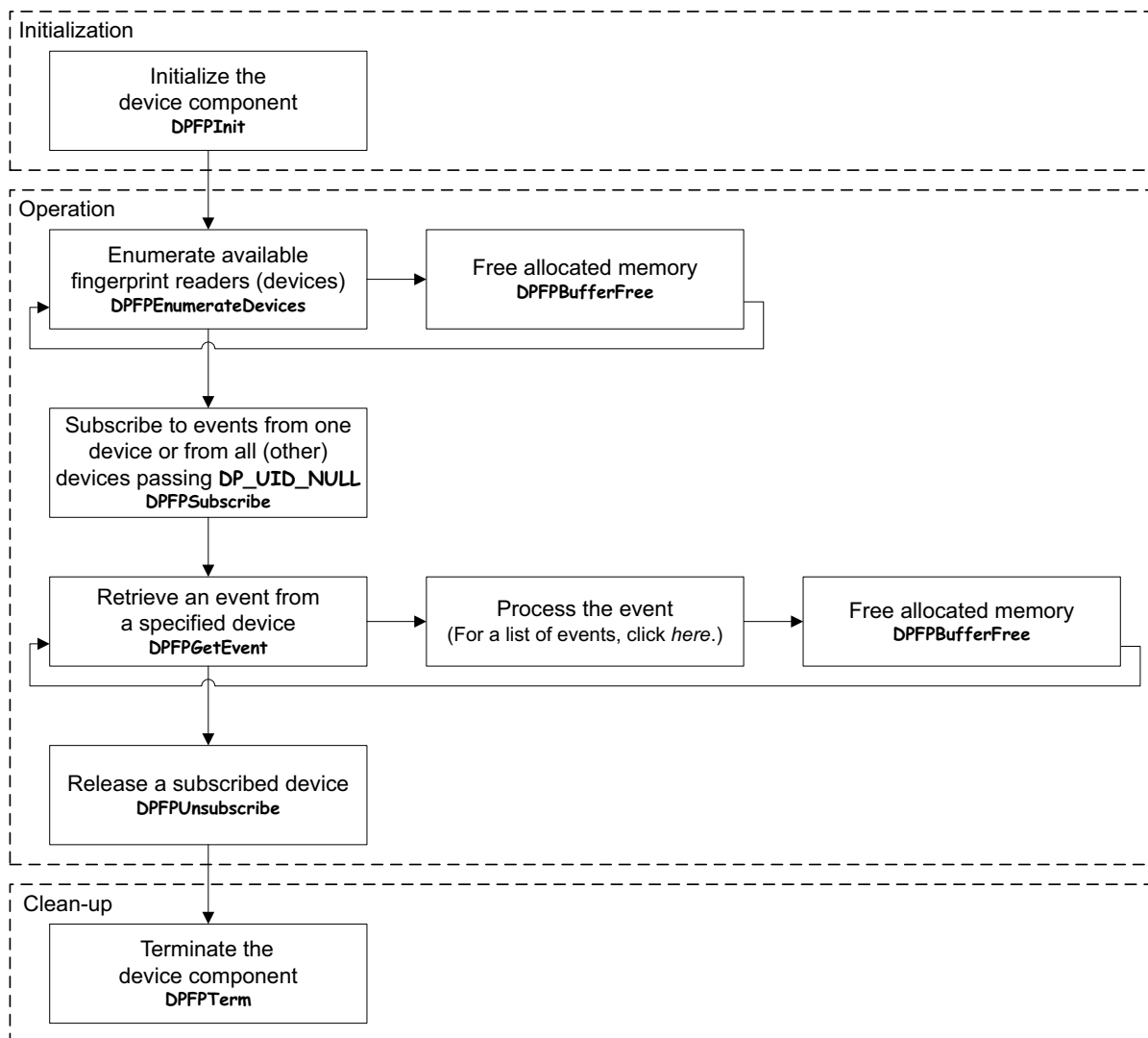


Figure 3. Device component workflow

Initialization

- Initialize the device component by calling the **DPFPInit** function (*page 35*).

Operation

Steps 1 and 2 may be repeated multiple times.

1. Enumerate the available fingerprint readers (devices) connected to a computer by calling the **DPFPEnumerateDevices** function (*page 31*).
2. Free the memory allocated by the **DPFPEnumerateDevices** function by calling the **DPFPBufferFree** function (*page 31*).
3. Subscribe to events from a single fingerprint reader once by calling the **DPFPSubscribe** function and passing the reader's UID. All subsequent calls to the **DPFPGetEvent** function retrieve events from that fingerprint reader only. You can also subscribe to all available fingerprint readers by passing the value **DP_UID_NULL**. All subsequent calls to the **DPFPGetEvent** function using this value retrieve events from all (other) readers.

Steps 4 through 6 may be repeated multiple times.

4. Retrieve an event from a specified fingerprint reader by calling the **DPFPGetEvent** function (*page 33*).
5. Process the event. The description of the **ppEvent** parameter used in the **DPFPGetEvent** function contains a list of events generated by the fingerprint reader (*page 34*).
6. Free the memory allocated by the **DPFPGetEvent** function by calling the **DPFPBufferFree** function (*page 31*).
7. Release a subscribed fingerprint reader by calling the **DPFPUnsubscribe** function (*page 37*).

Clean-up

- Terminate the device component when your application no longer requires access to any fingerprint readers by calling the **DPFPTerm** function (*page 36*).

Fingerprint Recognition Component

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

Fingerprint Enrollment

A *typical* fingerprint enrollment application workflow is represented in *Figure 4*, *Figure 5*, and *Figure 6*. Each figure is followed by explanations of the One Touch for Linux API functions that are used to perform the tasks in that part of the workflow. Both the fingerprint feature extraction and the fingerprint comparison modules are used for performing enrollment.

Typical Fingerprint Enrollment Workflow

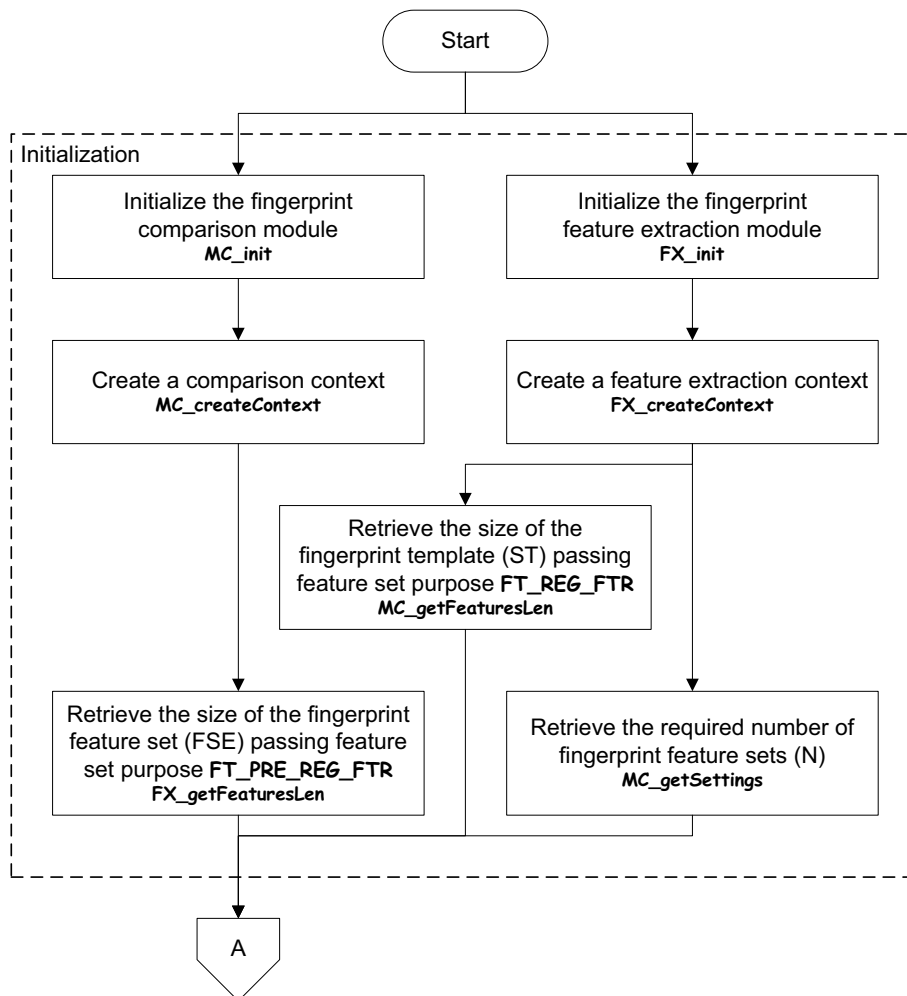


Figure 4. Typical fingerprint enrollment workflow: Initialization

Initialization Tasks

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

1. Initialize the fingerprint feature extraction module by calling the **FX_init** function (*page 45*).
2. Create a feature extraction *context* by calling the **FX_createContext** function (*page 38*).
3. Initialize the fingerprint comparison module by calling the **MC_init** function (*page 51*).
4. Create a comparison context by calling the **MC_createContext** function (*page 47*).

Steps 5 through 7 can be done in any order.

5. Retrieve the size of the fingerprint feature set (FSE) by calling the **FX_getFeaturesLen** function and passing feature set purpose **FT_PRE_REG_FTR** (*page 43*).
6. Retrieve the size of the fingerprint template (ST) by calling the **MC_getFeaturesLen** function and passing feature set purpose **FT_REG_FTR** (*page 49*).
7. Retrieve the number (N) of fingerprint feature sets required to create the fingerprint template by calling the **MC_getSettings** function (*page 50*).

Typical Fingerprint Enrollment Workflow (continued)

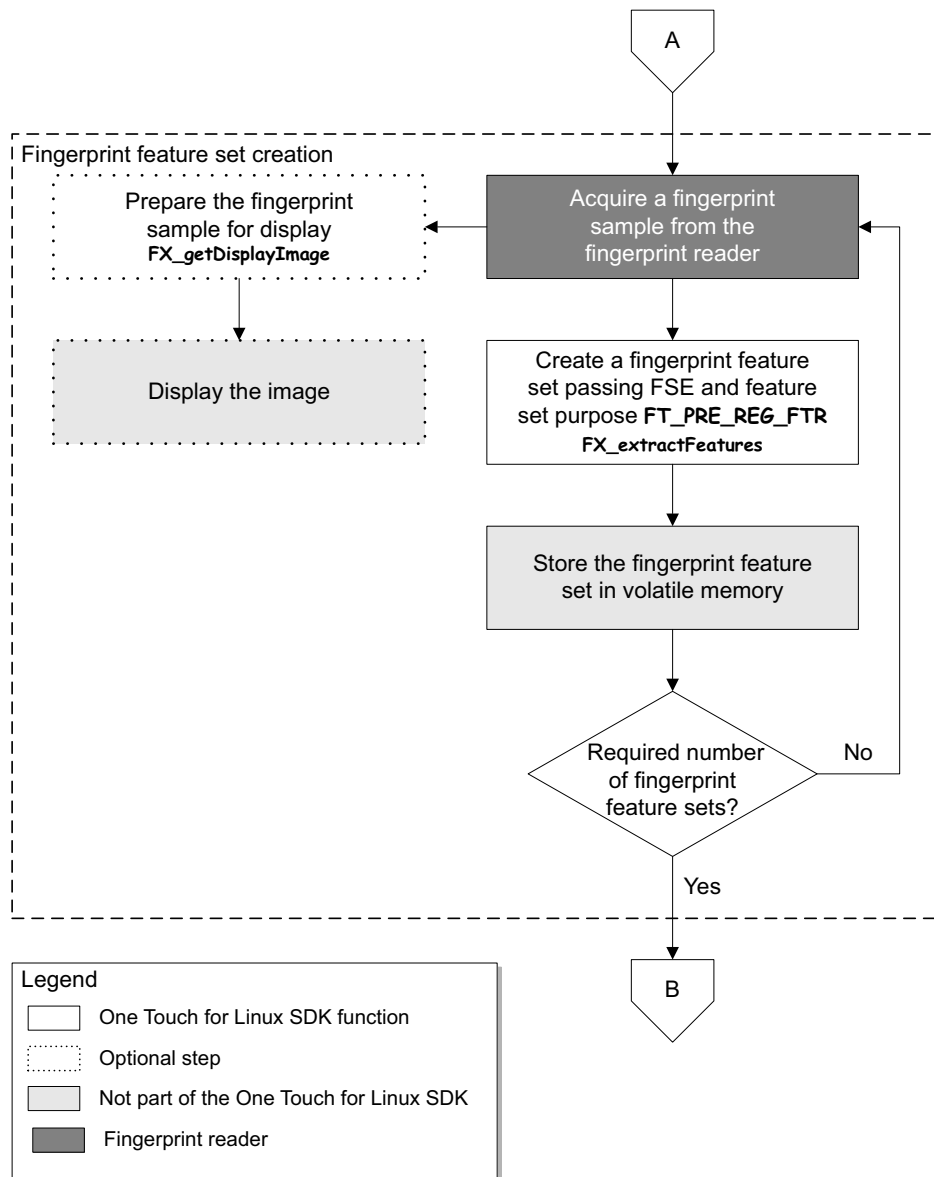


Figure 5. Typical fingerprint enrollment workflow: Fingerprint feature set creation

Fingerprint Feature Set Creation Tasks

Repeat the following required steps until you have created the number of fingerprint feature sets required to generate a fingerprint template. This number (N) was obtained when you called the **MC_getSettings** function (page 50) during initialization. (Steps preceded by an asterisk are not accomplished using One Touch for Linux API functions.)

1. *Acquire a fingerprint sample from the fingerprint reader.
2. Create a fingerprint feature set by calling the **FX_extractFeatures** function and passing FSE and feature set purpose **FT_PRE_REG_FTR** (page 39).

Steps 3 and 4 are optional.

3. Prepare the fingerprint sample acquired by the fingerprint reader for display by calling the **FX_getDisplayImage** function (page 42).
4. *Display the image.
5. *If the **FX_extractFeatures** function succeeds, store the resulting fingerprint feature set in volatile memory.

Typical Fingerprint Enrollment Workflow (*continued*)

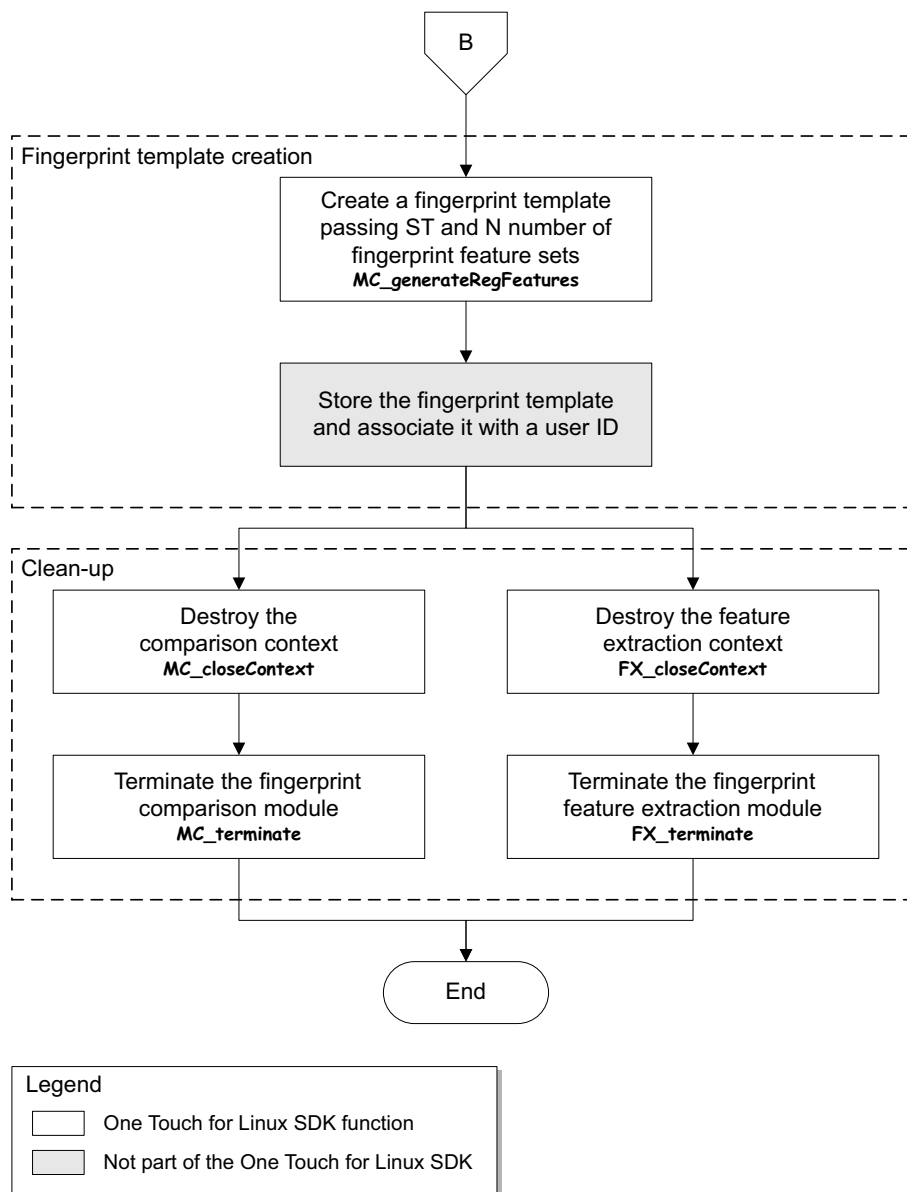


Figure 6. Typical fingerprint enrollment workflow: Fingerprint template creation and clean-up

Fingerprint Template Creation Tasks

1. Create a fingerprint template by calling the **MC_generateRegFeatures** function (*page 47*) and passing ST and the N number of fingerprint features sets created previously and stored in volatile memory.

Step 2 is not accomplished using One Touch for Linux API functions.

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

Clean-up Tasks

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

1. Destroy the comparison context by calling the **MC_closeContext** function (*page 46*)
2. Terminate the fingerprint comparison module by calling the **MC_terminate** function (*page 53*).
3. Destroy the feature extraction context by calling the **FX_closeContext** function (*page 38*)
4. Terminate the fingerprint feature extraction module by calling the **FX_terminate** function (*page 46*).

Fingerprint Verification

A *typical* fingerprint verification application workflow is represented in *Figure 7*, *Figure 8*, and *Figure 9*. Each figure is followed by explanations of the One Touch for Linux API functions that are used to perform the tasks in that part of the workflow. Both the fingerprint feature extraction and the fingerprint comparison modules are used for performing verification.

Typical Fingerprint Verification Workflow

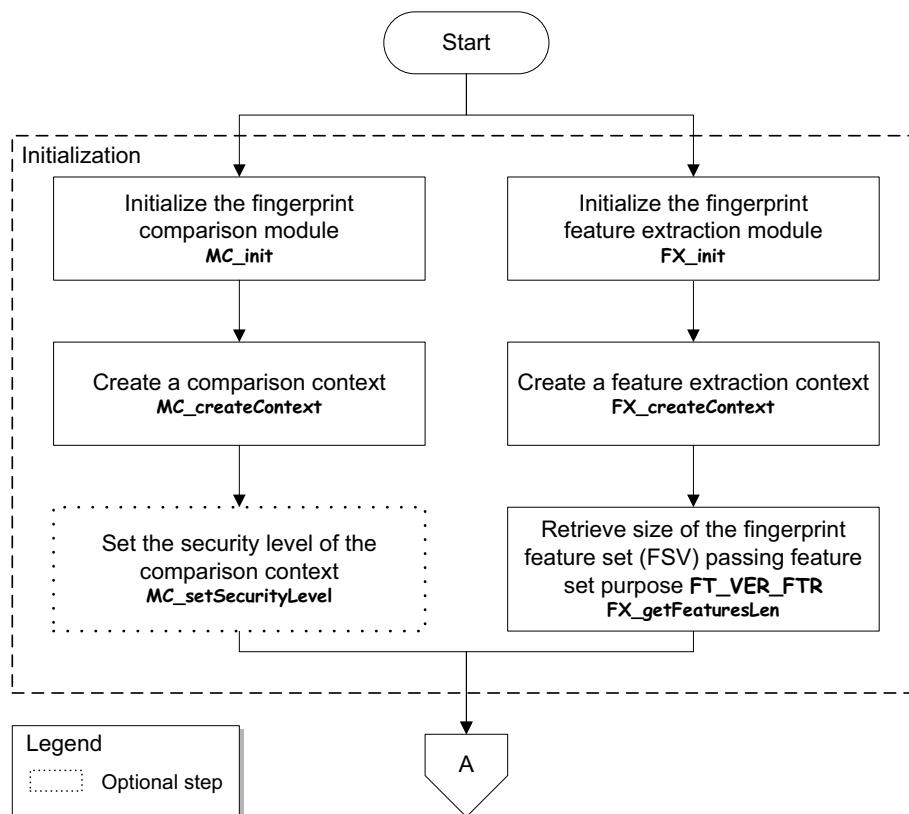
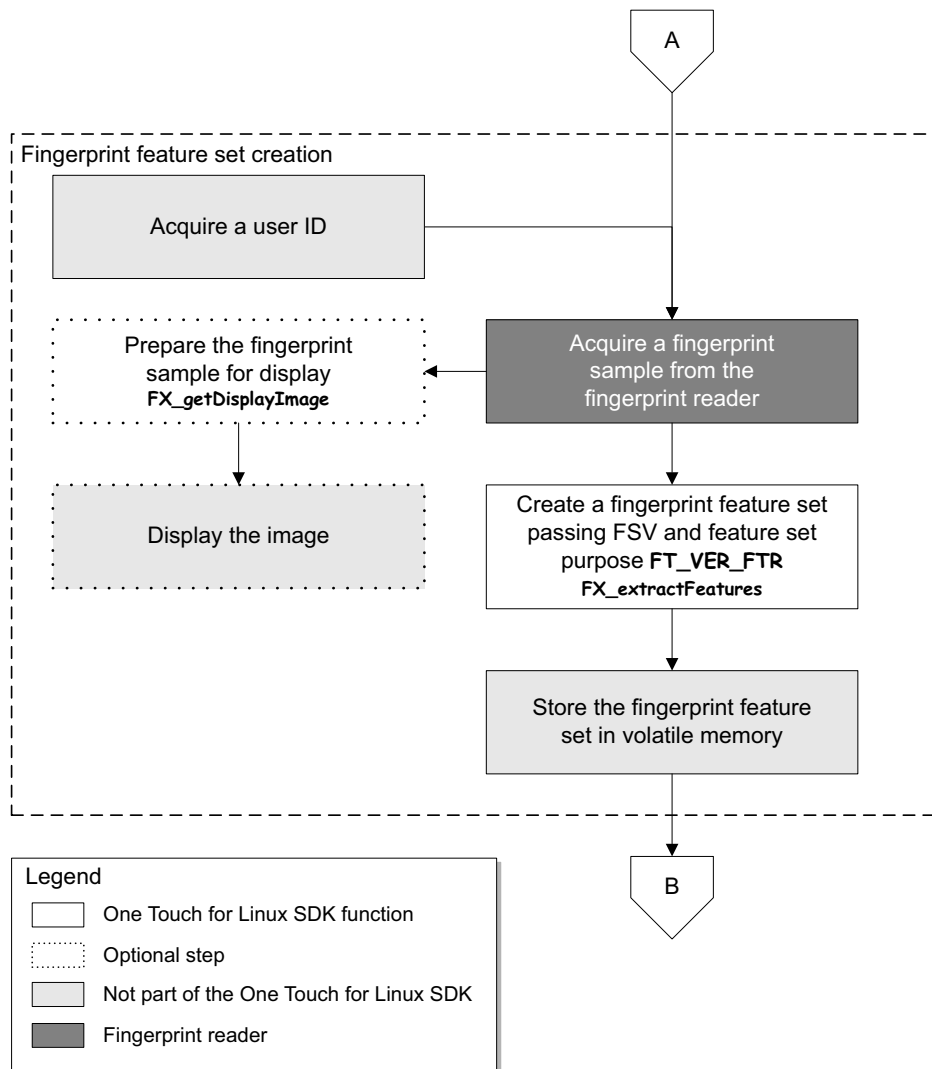


Figure 7. Typical fingerprint verification workflow: Initialization

Initialization Tasks

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

1. Initialize the fingerprint feature extraction module by calling the **FX_init** function (*page 45*).
2. Create a feature extraction context by calling the **FX_createContext** function (*page 38*).
3. Initialize the fingerprint comparison module by calling the **MC_init** function (*page 51*).
4. Create a comparison context by calling the **MC_createContext** function (*page 47*).
5. Optionally, set the security level of the comparison context by calling the **MC_setSecurityLevel** function (*page 52*). If you do not call this function, the default security level will be used.
6. Retrieve the size of the fingerprint feature set (FSV) by calling the **FX_getFeaturesLen** function and passing feature set purpose **FT_VER_FTR** (*page 43*).

Typical Fingerprint Verification Workflow (continued)**Figure 8.** Typical fingerprint verification workflow: Fingerprint feature set creation

Fingerprint Feature Set Creation Tasks

Steps preceded by an asterisk are not accomplished using One Touch for Linux API functions.

1. *Acquire the user ID that was used to associate the fingerprint template with the person to be verified.
2. *Acquire a fingerprint sample from the person via the fingerprint reader.
3. Create a fingerprint feature set by calling the **FX_extractFeatures** function and passing FSV and feature set purpose **FT_VER_FTR** (page 39).

Steps 4 and 5 are optional.

4. Prepare the fingerprint sample acquired by the fingerprint reader for display by calling the **FX_getDisplayImage** function (page 42).
5. *Display the image.
6. *If the **FX_extractFeatures** function succeeds, store the resulting fingerprint feature set in volatile memory.

Typical Fingerprint Verification Workflow (*continued*)

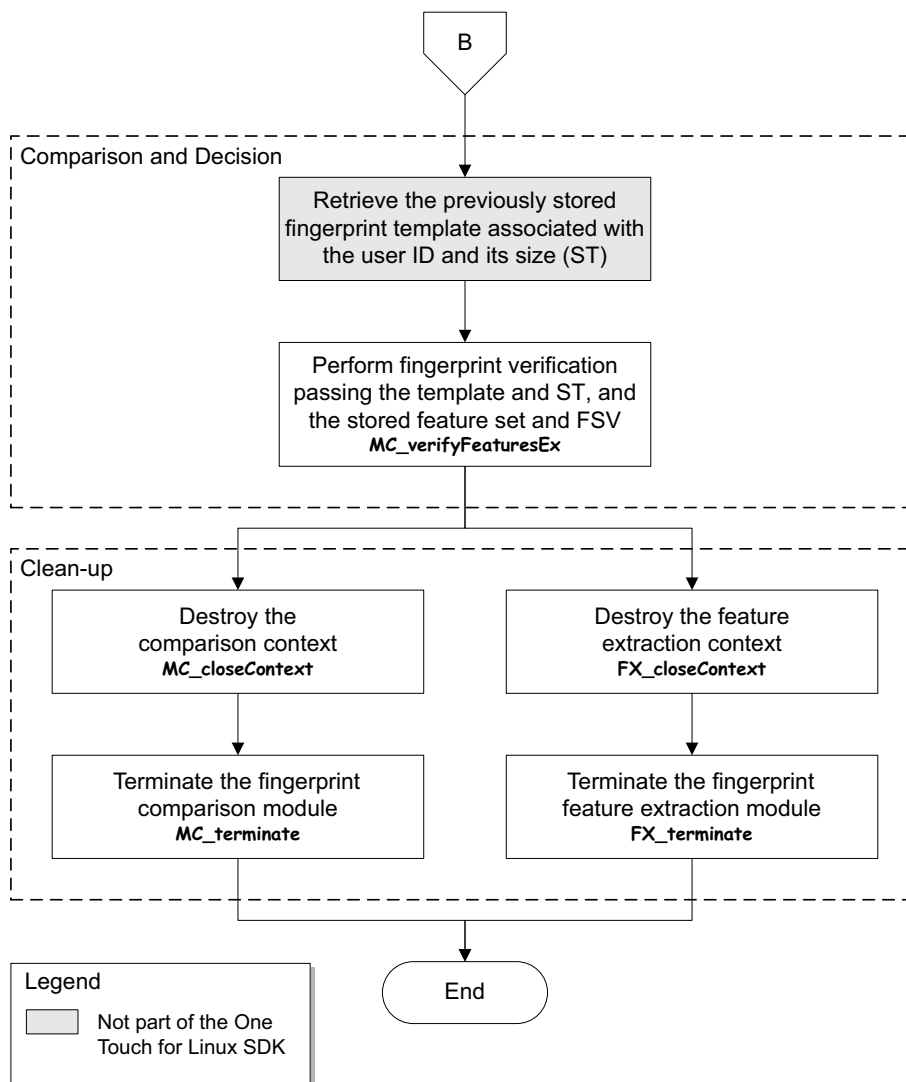


Figure 9. Typical fingerprint verification workflow: Comparison and decision, and clean-up

Comparison-and-Decision Tasks

Step 1 is not accomplished using One Touch for Linux API functions.

1. *Retrieve the fingerprint template associated with the user ID and size ST from your repository.
2. Perform fingerprint verification by calling the **MC_verifyFeaturesEx** function (*page 53*) and passing the stored fingerprint feature set together with FSV, and the fingerprint template together with ST.

Clean-up Tasks

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

1. Destroy the comparison context by calling the **MC_closeContext** function (*page 46*)
2. Terminate the fingerprint comparison module by calling the **MC_terminate** function (*page 53*).
3. Destroy the feature extraction context by calling the **FX_closeContext** function (*page 38*)
4. Terminate the fingerprint feature extraction module by calling the **FX_terminate** function (*page 46*).

This chapter defines the functions and data structures that are part of the One Touch for Linux API. The information in this chapter is extracted from the `dpFtrEx.h`, `dpMatch.h`, `dpDefs.h`, and `dpfp_api.h` files located in the `/opt/DigitalPersona/OneTouchSDK/include` directory.

Functions

This section defines the One Touch for Linux API functions. You can use the following list to quickly locate the functions contained in this section by function name, by page, or by description.

Function	Page	Description
Device Component		
DPFPBufferFree	31	Frees the memory allocated by the DPFPEnumerateDevices and DPFPGetEvent functions.
DPFPEnumerateDevices	31	Enumerates fingerprint readers (devices) connected to a computer.
DPFPGetDeviceInfo	32	Retrieves the information about a fingerprint reader.
DPFPGetEvent	33	Retrieves events from the specified fingerprint reader.
DPFPInit	35	Initializes the device component.
DPFPSubscribe	35	Subscribes to events from a fingerprint reader(s).
DPFPTerm	36	Terminates the device component.
DPFPUnsubscribe	37	Releases a subscribed fingerprint reader.
Fingerprint Feature Extraction Module		
FX_closeContext	38	Destroys a feature extraction context and releases the resources associated with it.
FX_createContext	38	Creates a context for the fingerprint feature extraction module.
FX_extractFeatures	39	Creates a fingerprint feature set for the purpose of fingerprint enrollment or verification.
FX_getDisplayImage	42	Prepares the fingerprint sample obtained from the fingerprint reader for display.
FX_getFeaturesLen	43	Retrieves the size of the buffer for the fingerprint feature set.
FX_getVersionInfo	44	Retrieves the software version information of the fingerprint feature extraction module.

Function	Page	Description
FX_init	45	Initializes the fingerprint feature extraction module.
FX_terminate	46	Terminates the fingerprint feature extraction module and releases the resources associated with it.
Fingerprint Comparison Module		
MC_closeContext	46	Destroys a comparison context and releases the resources associated with it.
MC_createContext	47	Creates a context for the fingerprint comparison module.
MC_generateRegFeatures	47	Creates a fingerprint template to be used for later comparison with a fingerprint feature set.
MC_getFeaturesLen	49	Retrieves the size of the buffer for the fingerprint template.
MC_getSecurityLevel	50	Retrieves the current security level of the specified comparison context in terms of the false accept rate (FAR).
MC_getSettings	50	Retrieves the current fingerprint comparison module settings.
MC_getVersionInfo	51	Retrieves the software version information of the fingerprint comparison module.
MC_init	51	Initializes the fingerprint comparison module.
MC_setSecurityLevel	52	Sets the security level of the specified comparison context by specifying a target false accept rate (FAR).
MC_terminate	53	Terminates the fingerprint comparison module and releases the resources associated with it.
MC_verifyFeaturesEx	53	Performs fingerprint verification.

DPFPBufferFree

Frees the memory allocated by the `DPFPEnumerateDevices` (page 31) and `DPFPGetEvent` (page 33) functions.

Syntax

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

Parameter Name

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

Library

libdpfpapi.so

DPFPEnumerateDevices

Enumerates fingerprint readers (devices) connected to a computer. Each fingerprint reader is represented by a device UID.

IMPORTANT: This function allocates memory, which must be freed by calling the `DPFPBufferFree` function (page 31).

Syntax

```
DPFP_STDAPI DPFPEnumerateDevices(  
    uint32_t* puDevCount,  
    dp_uid_t** ppDevUID  
);
```

Parameter Names

<code>puDevCount</code>	[out] Pointer to the variable receiving the total number of connected fingerprint readers
<code>ppDevUID</code>	[out] Pointer to the pointer of the array of device UIDs

Return Values

If the function succeeds, the value `0` is returned. If an error occurred, one of the following codes is returned:

DPFP_EUNKNOWN	An unexpected error occurred.
DPFP_ELIB_NOT_INITIALIZED	The device component library is not initialized.
EINVAL	Invalid parameter was passed to the function.
ENOMEM	There is not enough memory to perform the action.

Library

libdpfpapi.so

DPFPGetDeviceInfo

Retrieves the information about a fingerprint reader contained in the structure of type `dp_device_info_t` (page 57).

Syntax

```
DPFP_STDAPI DPFPGetDeviceInfo(  
    dp_uid_t* pDevUID,  
    dp_device_info_t* pDevInfo  
);
```

Parameter Names

pDevUID	[in] Pointer to the UID of the fingerprint reader to retrieve information about
pDevInfo	[out] Pointer to the memory receiving information about the fingerprint reader

Return Values

If the function succeeds, the value `0` is returned. If an error occurred, one of the following codes is returned:

DPFP_EUNKNOWN	An unexpected error occurred.
DPFP_ELIB_NOT_INITIALIZED	The device component library is not initialized.
DPFP_EDEV_INVALID_ID	There is no fingerprint reader with the specified device UID available.
EINVAL	Invalid parameter was passed to the function.
ENOMEM	There is not enough memory to perform the action.

Library

libdpfpapi.so

DPFPGetEvent

Retrieves events from the specified fingerprint reader. This is a blocking call. It will return if there is an event from the fingerprint reader, if the timeout has expired, or if the application unsubscribes from the fingerprint reader, that is, calls the **DPFPUnsubscribe** function (*page 37*).

IMPORTANT: This function allocates memory, which must be freed by calling the **DPFPBufferFree** function (*page 31*).

Syntax

```
DPFP_STDAPI DPFPGetEvent(  
    dp_uid_t* pDevUid,  
    dp_device_event_t** ppEvent,  
    uint32_t uTimeout  
);
```

Parameter Names

pDevUid	[in] Pointer to the UID of the fingerprint reader to retrieve an event from
ppEvent	<p>[out] Pointer to the pointer to the memory that receives information about the event. Events generated by the fingerprint reader are represented by the following values:</p> <p>DP_EVENT_DEVICE_CONNECTED. The fingerprint reader has been connected.</p> <p>DP_EVENT_DEVICE_DISCONNECTED. The fingerprint reader has been disconnected.</p> <p>DP_EVENT_FINGER_TOUCHED. A finger has touched the fingerprint reader.</p> <p>DP_EVENT_FINGER_GONE. A finger has been removed from the fingerprint reader.</p> <p>DP_EVENT_COMPLETED. A fingerprint sample has been captured by the fingerprint reader.</p> <p>DP_EVENT_STOPPED. If there is a blocking DPFPGetEvent function for the same UID on a different thread from which the DPFPUnsubscribe function (<i>page 37</i>) is called, this event is generated, and the function unblocks and returns DPFP_EDEV_NOT_SUBSCRIBED_ID for that device.</p>
uTimeout	[in] The uTimeout parameter is in milliseconds. If uTimeout is 0, the function returns immediately. If uTimeout is the value DP_TIMEOUT_INFINITE , the function returns only if the DPFPUnsubscribe function is called from a different thread.

Return Values

If the function succeeds, the value 0 is returned. If an error occurred, one of the following codes is returned:

DPFP_EUNKNOWN	An unexpected error occurred.
DPFP_ELIB_NOT_INITIALIZED	The device component library is not initialized.
DPFP_EDEV_SUBSCRIBED_NOT_ID	Events from the specified fingerprint reader are not subscribed to.
EINVAL	Invalid parameter was passed to the function.
ENOMEM	There is not enough memory to perform the action.
ETIMEDOUT	The timeout has expired.

Library

libdpfpapi.so

DPFPInit

Initializes the device component. This function must be called before calling any other function in the device component. Every successful call to this function should be paired with a call to the **DPFPTerm** function (page 36).

Syntax

```
DPFP_STDAP DPFPInit();
```

Return Values

If the function succeeds, the value `0` is returned. If an error occurred, one of the following codes is returned:

DPFP_EUNKNOWN	An unexpected error occurred.
DPFP_EDRV_NO_LIBRARY	The user mode library cannot be found.
DPFP_EDRV_NO_INTERFACE	The driver library did not export the expected interface.
DPFP_EDEVMGR_CANNOT_OPEN	An error occurred when opening the driver library.
ENOMEM	There is not enough memory to perform the action.

Library

libdpfpapi.so

See Also

DPFPTerm on page 36

DPFPSubscribe

Subscribes to events from a fingerprint reader(s). An application can subscribe to a fingerprint reader only once using its device UID, or to all available fingerprint readers using the value **DP_UID_NULL**. All subsequent calls to the **DPFPGetEvent** function using a specific device UID retrieve events from that fingerprint reader only. Calls to the **DPFPGetEvent** function using the value **DP_UID_NULL** retrieve events from all (other) connected fingerprint readers.

Syntax

```
DPFP_STDAPI DPFPSubscribe(  
    dp_uid_t* pDevUID,  
    dp_client_priority_t uReserved  
);
```


Parameter Names

pDevUID	[in] Pointer to the UID of a specific fingerprint reader. To subscribe to all (other) connected fingerprint readers, use the value DP_UID_NULL .
uReserved	[in] This parameter is reserved for future use and should always be set to the value DP_CLIENT_PRIORITY_NORMAL .

Return Values

If the function succeeds, the value **0** is returned. If an error occurred, one of the following codes is returned:

DPFP_EUNKNOWN	An unexpected error occurred.
DPFP_ELIB_NOT_INITIALIZED	The device component library is not initialized.
DPFP_EDEV_SUBSCRIBED_ID	Events from the specified fingerprint reader are already subscribed to.
DPFP_EDEV_INVALID_ID	There is no fingerprint reader with the specified device UID available.
EINVAL	Invalid parameter was passed to the function.
ENOMEM	There is not enough memory to perform the action.

Library

libdpfpapi.so

See Also

DPFPUnsubscribe on page 37

DPFPTerm

Terminates the device component. This function should be called to terminate the device component when the application no longer requires access to any fingerprint readers.

Syntax

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

Library

libdpfpapi.so

DPFPUnsubscribe

Releases a subscribed fingerprint reader. This function can be called from the same thread as the **DPFPSubscribe** function or from another thread. If there is a blocking **DPFPGetEvent** function for the same UID on a different thread from which the **DPFPUnsubscribe** function is called, the event **DP_EVENT_STOPPED** is generated (*page 33*), and the function unblocks and returns **DPFP_EDEV_NOT_SUBSCRIBED_ID** for that device.

Syntax

```
DPFP_STDAPI DPFPUnsubscribe(  
    dp_uid_t* pDevUID  
);
```

Parameter Name

pDevUID	[in] Pointer to the UID of the fingerprint reader to be unsubscribed from
----------------	---

Return Values

If the function succeeds, the value **0** is returned. If an error occurred, one of the following codes is returned:

DPFP_EUNKNOWN	An unexpected error occurred.
DPFP_ELIB_NOT_INITIALIZED	The device component library is not initialized.
DPFP_EDEV_SUBSCRIBED_NOT_ID	Events from the specified fingerprint reader are not subscribed to.
EINVAL	Invalid parameter was passed to the function.
ENOMEM	There is not enough memory to perform the action.

Library

libdpfpapi.so

See Also

DPFPSubscribe on *page 35*

FX_closeContext

Destroys a feature extraction context and releases the resources associated with it.

Syntax

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

Parameter Name

fxContext	[in] Handle to the feature extraction context
------------------	---

Return Values

FT_OK	The function succeeded.
FT_ERR_NO_INIT	The fingerprint feature extraction module is not initialized.
FT_ERR_INVALID_CONTEXT	The given feature extraction context is not valid.

Library

dpFtrEx.so

See Also

MC_createContext on page 47

FX_createContext

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

Syntax

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

Parameter Name

fxContext	[out] Pointer to the memory receiving the handle to the feature extraction context
------------------	--

Return Values

FT_OK	The function succeeded.
FT_ERR_NO_INIT	The fingerprint feature extraction module is not initialized.
FT_ERR_NO_MEMORY	There is not enough memory to create a feature extraction context.

Library

dpFtrEx.so

See Also

MC_closeContext on page 46

FX_extractFeatures

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

Syntax

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

Parameter Names

fxContext	[in] Handle to the feature extraction context
imageSize	[in] Size of the fingerprint sample obtained from the fingerprint reader, in bytes
imagePt	[in] Pointer to the buffer that contains the fingerprint sample obtained from the fingerprint reader
featureSetPurpose	[in] Feature set purpose. Specifies the purpose for which the fingerprint feature set is to be created. For a fingerprint feature set to be used for enrollment, use the value FT_PRE_REG_FTR ; for verification, use FT_VER_FTR . FT_REG_FTR is not a valid value for this function.
featureSetSize	[in] Fingerprint feature set size. This parameter is the size, in bytes, of the fingerprint feature set. Use the FX_getFeaturesLen function (page 43) to obtain information about which fingerprint feature set size to use.
featureSet	[out] Pointer to the location of the buffer receiving the fingerprint feature set
imageQualityPt	[out] Pointer to the buffer containing information about the quality of the fingerprint sample. Quality is represented by one of the following values: FT_GOOD_IMG . The fingerprint sample quality is good. FT_IMG_TOO_LIGHT . The fingerprint sample is too light. FT_IMG_TOO_DARK . The fingerprint sample is too dark. FT_IMG_TOO_NOISY . The fingerprint sample is too blurred. FT_LOW_CONTRAST . The fingerprint sample contrast is too low. FT_UNKNOWN_IMG_QUALITY . The fingerprint sample quality is undetermined.

featuresQualityPt	<p>[out] Pointer to the buffer containing information about the quality of the fingerprint features. If the fingerprint sample quality (imageQualityPt) is not equal to the value FT_GOOD_IMG, fingerprint feature extraction is not attempted, and this parameter is set to the value FT_UNKNOWN_FTR_QUALITY.</p> <p>Fingerprint features quality is represented by one of the following values:</p> <p>FT_GOOD_FTR. The fingerprint features quality is good.</p> <p>FT_NOT_ENOUGH_FTR. There are not enough fingerprint features.</p> <p>FT_NO_CENTRAL_REGION. The fingerprint sample does not contain the central portion of the finger.</p> <p>FT_AREA_TOO_SMALL. The fingerprint sample area is too small.</p> <p>FT_UNKNOWN_FTR_QUALITY. The fingerprint features quality is undetermined.</p>
featureSetCreated	<p>[out] Pointer to the memory receiving the value of whether the fingerprint feature set is created. If the value of this parameter is FT_TRUE, the fingerprint feature set was written to featureSet. If the value is FT_FALSE, a fingerprint feature set was not created.</p>

Return Values

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

Library

dpFtrEx.so

See Also

MC_generateRegFeatures on page 47

MC_verifyFeaturesEx on page 53

FX_getDisplayImage

Prepares the fingerprint sample obtained from the fingerprint reader for display. This may involve resizing, changing the number of grayscale intensity levels, rotating, and otherwise processing the fingerprint image to ensure that it displays well. The fingerprint sample passed to the **FX_getDisplayImage** function is the same fingerprint sample used by the **FX_extractFeatures** function ([page 39](#)).

Syntax

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

Parameter Names

fxContext	[in] Handle to the feature extraction context
imagePt	[in] Pointer to the buffer containing the fingerprint sample obtained from the fingerprint reader
pImageSize	[in] Pointer to the buffer containing the dimensions of the fingerprint sample obtained from the fingerprint reader
imageRotation	[in] Indicates whether the fingerprint image is to be rotated. If the value of this parameter is equal to FT_TRUE , the fingerprint image is rotated. If the value is FT_FALSE , the fingerprint image is not rotated.
numIntensityLevels	[in] Requested number of grayscale intensity levels
pImageBuffer	[in/out] Pointer to the buffer containing the fingerprint sample obtained from the fingerprint reader. The fingerprint image to be displayed is returned in the same buffer.

Return Values

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

Library

dpFtrEx.so

See Also

FX_extractFeatures on page 39

FX_getFeaturesLen

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

Syntax

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


Parameter Names

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

Return Values

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

Library

dpFtrEx.so

See Also

MC_generateRegFeatures on page 47

FX_getVersionInfo

Retrieves the software version information of the fingerprint feature extraction module in the structure of type **FT_VERSION_INFO** (page 59).

Syntax

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

Parameter Name

fxModuleVersionPt	[out] Pointer to the buffer containing the fingerprint feature extraction module software version information
--------------------------	---

Return Values

FR_OK	The function succeeded.
FR_ERR_BAD_INI_SETTING	Initialization settings are corrupted.

Library

dpFtrEx.so

FX_init

Initializes the fingerprint feature extraction module. This function must be called before any other function in the module is called.

Syntax

```
FX_DLL_INTERFACE FT_RETCODE FX_init(void);
```

Return Values

FR_OK	The function succeeded.
FR_ERR_NO_MEMORY	There is not enough memory to initialize the fingerprint feature extraction module.
FR_ERR_BAD_INI_SETTING	Initialization settings are corrupted.

Library

dpFtrEx.so

See Also

FX_terminate on page 46

FX_terminate

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

Syntax

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

Return Values

FR_OK	The function succeeded.
FR_WRN_NO_INIT	The fingerprint feature extraction module is not initialized.

Library

dpFtrEx.so

See Also

`FX_init` on page 45

MC_closeContext

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

Syntax

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

Parameter Name

mcContext	[in] Handle to the comparison module context
-----------	--

Library

dpMatch.so

See Also

`MC_closeContext` on page 46

MC_createContext

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

Syntax

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

Parameter Name

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

Library

dpMatch.so

See Also

MC_closeContext on [page 46](#)

MC_generateRegFeatures

Creates a fingerprint template to be used for later comparison with a fingerprint feature set. This function, known as *fingerprint enrollment*, computes the fingerprint template using the specified number of fingerprint feature sets (**numFeatureSets**) successfully returned by the **FX_extractFeatures** function ([page 39](#)).

Syntax

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

Parameter Names

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

Return Values

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

Library

dpMatch.so

See Also

FX_extractFeatures on page 39

MC_getFeaturesLen

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

Syntax

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

Parameter Names

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

Library

dpMatch.so

See Also

MC_generateRegFeatures on page 47

MC_getSecurityLevel

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

Syntax

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

Parameter Names

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

Library

dpMatch.so

See Also

MC_setSecurityLevel on page 52

MC_getSettings

Retrieves the current fingerprint comparison module settings in the structure of type **MC_SETTINGS** (page 59). This function provides the number of fingerprint feature sets required for the purpose of fingerprint enrollment. This setting is read-only.

Syntax

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

Parameter Name

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

Library

dpMatch.so

MC_getVersionInfo

Retrieves the software version information of the fingerprint comparison module in the structure of type **FT_VERSION_INFO** (page 59).

Syntax

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

Parameter Name

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

Library

dpMatch.so

MC_init

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

Syntax

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

Return Values

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

Library

dpMatch.so

See Also

MC_terminate on page 53

MC_setSecurityLevel

Sets the security level of a comparison context by specifying a target false accept rate (FAR). The lower the value of the FAR, the higher the security level and the higher the target false reject rate (FRR). (See *False Positives and False Negatives* on page 11 for more information about FAR and FRR.)

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

Syntax

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

Parameter Names

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

Return Values

FR_OK	The function is successful
FR_ERR_NO_INIT	The fingerprint comparison module is not initialized.
FR_ERR_INVALID_PARAM	The value of the parameter targetFar ≤ 0.0 or ≥ 100.0 , or the specified comparison context is not valid.
FR_ERR_INVALID_CONTEXT	The specified comparison context is not valid.
FR_WRN_INTERNAL	The value of the parameter targetFar is unacceptably high and was reduced to an internally defined value.

Library

dpMatch.so

See Also

MC_getSecurityLevel on page 50

MC_terminate

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

Syntax

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

Library

dpMatch.so

See Also

`MC_init` on page 51

MC_verifyFeaturesEx

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

Syntax

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

Parameter Names

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

Return Values

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

Library

dpMatch.so

See Also

MC_generateRegFeatures on *page 47*

Data Structures

This section defines the One Touch for Linux API data structures.

dp_device_event_t

Event type structure.

Data Fields

dUint32_t nEvent	<p>Type of event. Events generated by the fingerprint capture device are represented by the following values:</p> <p>DP_EVENT_DEVICE_CONNECTED. The device has been connected.</p> <p>DP_EVENT_DEVICE_DISCONNECTED. The device has been disconnected.</p> <p>DP_EVENT_FINGER_TOUCHED. A finger has touched the device.</p> <p>DP_EVENT_FINGER_GONE. A finger has been removed from the device.</p> <p>DP_EVENT_COMPLETED. A fingerprint sample has been captured by the sensor on the device.</p> <p>DP_EVENT_STOPPED. If there is a blocking DPFPGetEvent function (<i>page 33</i>) for the same UID on a different thread from which the DPFPUnsubscribe function (<i>page 37</i>) is called, this event is generated, and the function unblocks and returns DPFP_EDEV_NOT_SUBSCRIBED_ID for that device.</p>
dp_uid_t DeviceUid	Unique identifier of the fingerprint capture device assigned by the driver.
uint32_t uDataSize	Size of the data associated with the event. The size of this field is 0 for all events except DP_EVENT_COMPLETED , which is the size of the actual fingerprint feature set (approximately 300 bytes).
uint8_t Data[1]	Actual data associated with the event, which is used for DP_EVENT_COMPLETED only.

dp_device_info_t

Fingerprint capture device information structure.

Data Fields

<code>dp_uid_t DeviceUid</code>	Unique identifier of the fingerprint capture device assigned by the driver.
<code>dp_device_uid_type_t eUidType</code>	Indicates whether the device UID persists after reboot, as represented by the following values: DP_PERSISTENT_DEVICE_UID. Persistent UIDs. (DigitalPersona fingerprint readers have persistent UIDs.) DP_VOLATILE_DEVICE_UID. Volatile UIDs.
<code>dp_device_modality_t eDeviceModality</code>	Indicates the modality that the device uses to capture fingerprint samples, as represented by the following values: DP_AREA_DEVICE. Area (touch) sensor devices. (DigitalPersona fingerprint readers are area sensor devices.) DP_SWIPE_DEVICE. Swipe devices. DP_UNKNOWN_DEVICE_MODALITY. Devices for which the modality is not known.
<code>dp_device_technology_t eDeviceTech</code>	Indicates the fingerprint capture device technology, as represented by the following values: DP_OPTICAL_DEVICE. Optical devices. (DigitalPersona fingerprint readers are 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.
<code>dp_hw_info_t HwInfo</code>	Fingerprint capture device information returned by the driver, which is contained in the structure of type <code>dp_hw_info_t</code> .

dp_hw_info_t

Fingerprint capture device information structure returned by the driver.

Data Fields

<code>uint32_t uLanguageId</code>	This field is always 0x409.
<code>char szVendor[DP_MAX_STRING_SIZE]</code>	Fingerprint capture device vendor name, for example, "DigitalPersona, Inc."
<code>char szProduct[DP_MAX_STRING_SIZE]</code>	Fingerprint capture device product name, for example, "U.are.U"
<code>char szSerialNb[DP_MAX_STRING_SIZE]</code>	Serial number of the fingerprint capture device, which is converted to the device UID by the driver, for example "{7C265680-0056-FFFF-680D-A74033B09615}"
<code>dp_version_t HardwareRevision</code>	Hardware revision number of the fingerprint capture device
<code>dp_version_t FirmwareRevision</code>	Firmware revision number of the fingerprint capture device

dp_version_t

Version structure of the hardware and firmware of the fingerprint capture device.

Data Fields

<code>uint32_t uMajor</code>	Major version number
<code>uint32_t uMinor</code>	Minor version number
<code>uint32_t uRevision</code>	Revision number
<code>uint32_t uBuild</code>	Build number

MC_SETTINGS

Fingerprint comparison module settings structure.

Data Field

<code>int numFeatureSets</code>	Number of fingerprint feature sets required to generate a fingerprint template
---------------------------------	--

FT_VERSION_INFO

Fingerprint feature extraction or fingerprint comparison module version information structure.

Data Fields

<code>unsigned major</code>	Major version number
<code>unsigned minor</code>	Minor version number
<code>unsigned revision</code>	Revision number
<code>unsigned build</code>	Build number

This chapter defines the return codes used in the One Touch for Linux API.

Device Component

The following table is extracted from the `dpfp_api_errors.h` file located in the `/opt/DigitalPersona/OneTouchSDK/include` directory.

Return Code	Description
<code>DPFP_EDEV_INVALID_ID</code>	There is no device with the specified UID available.
<code>DPFP_EDEV_NOT_SUBSCRIBED_ID</code>	Events from the specified device are not subscribed to.
<code>DPFP_EDEV_SUBSCRIBED_ID</code>	Events from the specified device are already subscribed to.
<code>DPFP_EDEVMGR_CANNOT_OPEN</code>	An error occurred when opening the driver library.
<code>DPFP_EDRV_NO_INTERFACE</code>	The driver library did not export the expected interface.
<code>DPFP_EDRV_NO_LIBRARY</code>	The user mode library cannot be found.
<code>DPFP_ELIB_NOT_INITIALIZED</code>	The device component library is not initialized.
<code>DPFP_EUNKNOWN</code>	An unexpected error occurred.
<code>ENOMEM</code>	There is not enough memory to perform the action. (From <code>errno.h</code> .)
<code>ETIMEDOUT</code>	The timeout has expired. (From <code>errno.h</code> .)

Fingerprint Recognition Component

The following table is extracted from the `dpRCodes.h` located file in the `/opt/DigitalPersona/OneTouchSDK/include` directory.

Return Code	Description
<code>FT_ERR_BAD_INI_SETTING</code>	Initialization settings are corrupted.
<code>FT_ERR_FEAT_LEN_TOO_SHORT</code>	The specified fingerprint feature set or fingerprint template buffer size is too small.
<code>FT_ERR_FTRS_INVALID</code>	Decrypted fingerprint features are not valid. Decryption may have failed.

Return Code	Description
FT_ERR_INTERNAL	An unknown internal error occurred.
FT_ERR_INVALID_BUFFER	A buffer is not valid.
FT_ERR_INVALID_CONTEXT	The given context is not valid.
FT_ERR_INVALID_FTRS_TYPE	The feature set purpose is not valid.
FT_ERR_INVALID_PARAM	One or more parameters are not valid.
FT_ERR_IO	A generic I/O file error occurred.
FT_ERR_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module is not initialized.
FT_ERR_NO_MEMORY	There is not enough memory to perform the action.
FT_ERR_NOT_IMPLEMENTED	The called function was not implemented
FT_ERR_UNKNOWN_DEVICE	The fingerprint reader is not known.
FT_OK	The function succeeded.
FT_WRN_INTERNAL	An internal error occurred.
FT_WRN_KEY_NOT_FOUND	The fingerprint feature extraction module or the fingerprint comparison module could not find an initialization setting.
FT_WRN_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module are not initialized.
FT_WRN_UNKNOWN_DEVICE	The fingerprint reader is not known.

You may redistribute the files in the `redist` directory in the One Touch for Linux SDK product package to your end users pursuant to the terms of the End User License Agreement (EULA) located in the `/docs` directory in the product package.

When you develop a product based on the One Touch for Linux SDK, you must provide device component, fingerprint recognition component, and driver libraries to your end users. Collectively, these libraries and files constitute the One Touch for Linux SDK Runtime Environment (RTE).

Use the installer package that corresponds to your Linux kernel version. Also see the RTE installation instructions in the installation guide for the distribution/kernel version you are using.

Device Component Libraries

The device component libraries are located in the `/redistr` directory in the One Touch for Linux SDK product package.

- `DigitalPersona-fpapi-1.1.0-1.i586.rpm`

RPM package installer, which contains the following files:

- `libdpfpapi.so.1.1.0`
- `libdpfpapi.so -> libdpfpapi.so.1`

- `DigitalPersona-fpapi-1.1.0-1.i586.tar.gz`

TAR package installer, which contains the following files:

- `libdpfpapi.so.1.1.0`
- `libdpfpapi.so -> libdpfpapi.so.1`

Fingerprint Recognition Component Libraries

The fingerprint recognition component libraries are located in the `/redistr` directory in the One Touch for Linux SDK product package.

- `DigitalPersona-fprec-1.1.0-1.i586.rpm`

RPM package installer, which contains the following files:

- `libdpFtrEx.so.4.1.2`
- `libdpFtrEx.so -> libdpFtrEx.so.4`
- `libdpMatch.so.4.1.2`
- `libdpMatch.so -> libdpMatch.so.4`

- fpToolkit.reg
- DigitalPersona-fprec-1.1.0-1.i586.tar.gz

TAR package installer, which contains the following files:

- libdpFtrEx.so.4.1.2
- libdpFtrEx.so -> libdpFtrEx.so.4
- libdpMatch.so.4.1.2
- libdpMatch.so -> libdpMatch.so.4
- fpToolkit.reg

User Mode Driver Libraries

The user mode driver libraries are located in the /redistr directory in the One Touch for Linux SDK product package.

- DigitalPersona-fpusdrv-1.1.0-1.i586.rpm

RPM package installer, which contains the following files:

- libdpD00701.so.1.1.0
- libdpD00701.so -> libdpD00701.so.1
- libdpDevMgr.so.1.1.0
- libdpDevMgr.so -> libdpDevMgr.so.1
- libdpDrvApi.so.1.1.0
- libdpDrvApi.so -> libdpDrvApi.so.1
- libdpDrvDatApi.so.1.1.0
- libdpDrvDatApi.so -> libdpDrvDatApi.so.1
- libdpFC.so.1.1.0
- libdpFC.so -> libdpFC.so.1
- libdpI00701.so.1.1.0
- libdpI00701.so -> libdpI00701.so.1
- libdpObjMgr.so.1.1.0
- libdpObjMgr.so -> libdpObjMgr.so.1
- libdpUsbAda.so.1.1.0
- libdpUsbAda.so -> libdpUsbAda.so.1

- DigitalPersona-fpusrdrv-1.1.0-1.i586.tar.gz

TAR package installer, which contains the following files:

- libdpD00701.so.1.1.0
- libdpD00701.so -> libdpD00701.so.1
- libdpDevMgr.so.1.1.0
- libdpDevMgr.so -> libdpDevMgr.so.1
- libdpDrvApi.so.1.1.0
- libdpDrvApi.so -> libdpDrvApi.so.1
- libdpDrvDatApi.so.1.1.0
- libdpDrvDatApi.so -> libdpDrvDatApi.so.1
- libdpFC.so.1.1.0
- libdpFC.so -> libdpFC.so.1
- libdpl00701.so.1.1.0
- libdpl00701.so -> libdpl00701.so.1
- libdpObjMgr.so.1.1.0
- libdpObjMgr.so -> libdpObjMgr.so.1
- libdpUsbAda.so.1.1.0
- libdpUsbAda.so -> libdpUsbAda.so.1

Kernel Mode Driver Source Files

The kernel mode driver source files are located in the /redistr directory in the One Touch for Linux SDK product package.

- DigitalPersona-fpkrndrv-source-1.1.0-1.i586.rpm

RPM package installer for kernel mode driver source files, which contains the following files in the /source directory for kernel version 2.6 and in the /source_2.4 directory for kernel version 2.4:

- /source directory
 - Makefile
 - usbdpfp.c
 - usbdpfp.h
 - usbdpfpi.h

- /source_2.4 directory

- Makefile
- usbdpfp.c
- usbdpfp.h
- usbdpfp.h

- DigitalPersona-fpkrndrv-source-1.1.0-1.i586.tar.gz

TAR package installer for kernel mode driver source files, which contains the following files in the /source directory for kernel version 2.6 and in the /source_2.4 directory for kernel version 2.4:

- /source directory

- Makefile
- usbdpfp.c
- usbdpfp.h
- usbdpfp.h

- /source_2.4 directory

- Makefile
- usbdpfp.c
- usbdpfp.h
- usbdpfp.h

Kernel Mode Driver Files

The One Touch for Linux SDK product package contains a /redistr/drivers/<kernelversion> directory for each supported kernel version, where <kernelversion> is the Linux kernel version string, for example, 2.6.5-7.155.29-default. Each directory contains the following files:

- DigitalPersona-fpkrndrv-1.1.0-1.i586.rpm

RPM package installer for kernel mode driver files, which contains the following files:

- mod_usbdpfp.ko (for 2.6 kernels) or mod_usbdpfp.o (for 2.4 kernels)
- hotplug.sh

- DigitalPersona-fpkrndrv-1.1.0-1.i586.tar.gz

TAR package installer for kernel mode driver files, which contains the following files:

- mod_usbdpfp.ko (for 2.6 kernels) or mod_usbdpfp.o (for 2.4 kernels)
- hotplug.sh

Fingerprint Reader Documentation

You may redistribute the documentation included in the /redistr directory in the One Touch for Linux SDK product package to your end users pursuant to the terms of the EULA, which is located in the /docs directory in the product package, and of this section.

Hardware Warnings and Regulatory Information

If you distribute DigitalPersona U.are.U fingerprint readers to your end users, you are responsible for advising them of the warnings and regulatory information included in the Warnings_and_Regulatory_Information.pdf file in the /redistr directory in the One Touch for Linux SDK product package. You may copy and redistribute the language, including the copyright and trademark notices, set forth in the Warnings_and_Regulatory_Information.pdf file.

Fingerprint Reader Use and Maintenance Guide

The DigitalPersona U.are.U Fingerprint Reader Use and Maintenance Guide, DigitalPersona_Reader_Maintenance.pdf, is located in the /redistr directory in the One Touch for Linux SDK product package. You may copy and redistribute the DigitalPersona_Reader_Maintenance.pdf file, including the copyright and trademark notices, to those who purchase a U.are.U module or fingerprint reader from you.

Glossary

biometric system

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

comparison

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

comparison score

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

context

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

DigitalPersona Fingerprint Recognition Engine

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

enrollee

See **fingerprint data subject**.

enrollment

See **fingerprint enrollment**.

false accept rate (FAR)

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

false reject rate (FRR)

The proportion of fingerprint verification transactions by fingerprint enrollment subjects

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

features

See **fingerprint features**.

fingerprint

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

fingerprint capture device

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

fingerprint characteristic

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

fingerprint data

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

fingerprint data storage subsystem

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

fingerprint data subject

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

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

fingerprint enrollment

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

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

fingerprint enrollment subject

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

fingerprint feature extraction

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

fingerprint features

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

fingerprint characteristics.

fingerprint feature set

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

fingerprint image

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

fingerprint reader

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

fingerprint recognition system

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

fingerprint sample

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

fingerprint template

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

fingerprint verification

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

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

knowledge-based system

A method of identifying a person based on something that the person "knows," such as a password or an account number.

match

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

non-match

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

one-to-one comparison

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

repository

See **fingerprint data storage subsystem**.

security level

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

token-based system

A method of identifying a person based on something that the person “has,” such as a card or a key.

verification

See **fingerprint verification**.

Index

A

- additional resources 3
 - online resources 4
 - related documentation 3
- API
 - See One Touch for Linux API
- audience for developer guide 1

B

- biometric system
 - defined 67
 - explained 9
- bold typeface, uses of 3

C

- chapters, overview of 1
- comparison context
 - creating
 - function for 47
 - in typical enrollment workflow 18
 - in typical verification workflow 24
 - destroying
 - function for 46
 - in typical enrollment workflow 22
 - in typical verification workflow 28
- comparison module
 - data structures defined 59
 - functions defined 46–55
 - purpose of 12
- comparison module library
 - redistributing 62
- comparison, defined 67
- concepts and terminology 9
- context 18
 - creating
 - comparison context
 - function for 47
 - in typical enrollment workflow 18
 - in typical verification workflow 24
 - feature extraction context
 - function for 38
 - in typical enrollment workflow 18
 - in typical verification workflow 24
 - defined 67
 - destroying
 - comparison context
 - function for 46
 - in typical enrollment workflow 22

- in typical verification workflow 28
- feature extraction context
 - function for 38
 - in typical enrollment workflow 22
 - in typical verification workflow 28
- conventions, document
 - See document conventions
- Courier bold typeface, use of 3
- creating
 - comparison context
 - function for 47
 - in typical enrollment workflow 18
 - in typical verification workflow 24
 - feature extraction context
 - function for 38
 - in typical enrollment workflow 18
 - in typical verification workflow 24
- fingerprint feature set
 - function for 39
 - in typical enrollment workflow 20
 - in typical verification workflow 26
- fingerprint template
 - function for 47
 - in typical enrollment workflow 22

D

- data flow, SDK 12
 - illustrated 13
- data structures, One Touch for Linux API
 - defined 56
 - See also individual data structures by name
- destroying
 - comparison context
 - function for 46
 - in typical enrollment workflow 22
 - in typical verification workflow 28
 - feature extraction context
 - function for 38
 - in typical enrollment workflow 22
 - in typical verification workflow 28
- developer guide
 - document conventions 2
 - introduction to 1
 - overview of chapters 1
 - target audience for 1
 - terminology and concepts 9

- device
 - See fingerprint reader
- device component
 - data structures defined 56–58
 - functions defined 31–37
 - library
 - redistributing 62
 - purpose of 12
 - return codes 60
 - workflow 14–15
 - illustrated 14
 - tasks 15
- DigitalPersona Developer Connection Forum, URL to 4
- DigitalPersona fingerprint reader
 - See fingerprint reader
- DigitalPersona Fingerprint Recognition Engine 9
- DigitalPersona fingerprint recognition system 10
 - illustrated 11
- DigitalPersona products, supported 4
- document conventions 2
 - naming 2
 - notational 2
 - typographical 3
- documentation, related 3
- dp_device_event_t data structure, defined 56
- dp_device_info_t data structure, defined 57
- dp_hw_info_t data structure, defined 58
- DP_UID_NULL value
 - defined 36
 - using in device component workflow 15
- dp_version_t data structure, defined 58
- DPFPBufferFree function
 - defined 31
 - using in device component workflow 15
- DPFPEnumerateDevices function
 - defined 31
 - using in device component workflow 15
- DPFPGetDeviceInfo function, defined 32
- DPFPGetEvent function
 - defined 33
 - using in device component workflow 15
- DPFPInit function
 - defined 35
 - using in device component workflow 15
- DPFPSubscribe function
 - defined 35
 - using in device component workflow 15
- DPFPTerm function
 - defined 36
 - using in device component workflow 15

- DPFPUnsubscribe function
 - defined 37
 - using in device component workflow 15
- driver 10
 - kernel mode
 - redistributing 65
 - redistributing source files 64
 - user mode
 - redistributing libraries 63

E

- End User License Agreement
 - redistribution of One Touch for Linux SDK libraries 62
- Engine
 - See DigitalPersona Fingerprint Recognition Engine
- enrollee 10
 - defined 67
- enrollment
 - See fingerprint enrollment
- EULA
 - See End User License Agreement
- events from fingerprint reader
 - defined 34
 - retrieving
 - function for 33
 - in device component workflow 15
 - subscribing to
 - function for 35
 - in device component workflow 15
 - unsubscribing from
 - function for 37
 - in device component workflow 15

F

- false accept rate 12
 - defined 67
 - setting target for comparison context
 - function for 52
 - in typical verification workflow 24
- false negative decision 11
- false negative decision, proportion of 12
 - See also false accept rate
- false positive decision 11
- false positive decision, proportion of 12
 - See also false accept rate
- false positives and false negatives 11
- false reject rate 12
 - defined 67
- FAR
 - See false accept rate

- feature extraction context
 - creating
 - function for 38
 - in typical enrollment workflow 18
 - in typical verification workflow 24
 - destroying
 - function for 38
 - in typical enrollment workflow 22
 - in typical verification workflow 28
 - important notice not to set security level for 52
- feature extraction library
 - redistributing 62
- feature extraction module
 - data structures defined 59
 - functions defined 38–46
 - purpose of 12
- features
 - See fingerprint features
- fingerprint 9
 - defined 67
- fingerprint capture device 10
 - defined 67
 - See fingerprint reader
- fingerprint characteristics, defined 67
- fingerprint comparison module
 - See comparison module
- fingerprint data 10
 - defined 67
- fingerprint data storage subsystem, defined 67
- fingerprint data subject
 - defined 67
- fingerprint enrollment 10
 - defined 68
 - typical workflow 16–22
 - clean-up
 - illustrated 21
 - tasks 22
 - fingerprint feature set creation
 - illustrated 19
 - tasks 20
 - fingerprint template creation
 - illustrated 21
 - tasks 22
 - initialization
 - illustrated 17
 - tasks 18
- fingerprint feature extraction
 - defined 68
 - performing
 - function for 39
 - in typical enrollment workflow 20, 26
- fingerprint feature extraction module
 - See feature extraction module
- fingerprint feature set 10
 - creating
 - function for 39
 - in typical enrollment workflow 20
 - in typical verification workflow 26
 - defined 68
 - retrieving number required for fingerprint template
 - creation
 - function for 50
 - in typical enrollment workflow 18
 - retrieving size of
 - function for 43
 - in typical enrollment workflow 18
 - in typical verification workflow 24
- fingerprint features, defined 68
- fingerprint image
 - defined 68
 - preparing for display
 - function for 42
 - in typical enrollment workflow 20
 - in typical verification workflow 26
 - See also fingerprint sample
- fingerprint reader 10
 - defined 68
 - enumerating
 - function for 31
 - in device component workflow 15
 - redistributing documentation for 66
 - retrieving events from
 - function for 33
 - in device component workflow 15
 - retrieving information about, function for 32
 - subscribing to events from
 - function for 35
 - in device component workflow 15
 - supported 4
 - unsubscribing from events from
 - function for 37
 - in device component workflow 15
 - use and maintenance guide, redistributing 66
- fingerprint recognition 10
 - guide to
 - related documentation 3

- fingerprint recognition component 16
 - data structures, defined 59
 - functions, defined 38–55
 - libraries, redistributing 62
 - purpose of 12
 - return codes 60
- fingerprint recognition system 9
 - defined 68
 - See also* DigitalPersona fingerprint recognition system
- fingerprint sample
 - defined 68
 - preparing for display
 - function for 42
 - in typical enrollment workflow 20
 - in typical verification workflow 26
 - See also* fingerprint image
- fingerprint template 10
 - creating
 - function for 47
 - in typical enrollment workflow 22
 - defined 68
 - retrieving size of
 - function for 49
 - in typical enrollment workflow 18
- fingerprint verification 10
 - defined 68
 - performing
 - function for 53
 - in typical verification workflow 28
- typical workflow 23–28
 - clean-up
 - illustrated 27
 - tasks 28
 - comparison and decision
 - illustrated 27
 - tasks 28
 - fingerprint feature set creation
 - illustrated 25
 - tasks 26
 - initialization
 - illustrated 23
 - tasks 24
- freeing memory
 - allocated by DPFPEnumerateDevices and DPFPGetEvent functions, function for 31
 - allocated by DPFPEnumerateDevices function, in device component workflow 15
 - allocated by DPFPGetEvent function, in device component workflow 15
- FRR
 - See* false reject rate
- FT_PRE_REG_FTR value
 - defined 40
 - using in typical enrollment workflow 18, 20
- FT_REG_FTR value
 - defined 49
 - using in typical enrollment workflow 18
- FT_VER_FTR value
 - defined 40
 - using in typical verification workflow 24, 26
- FT_VERSION_INFO structure, defined 59
- functions, One Touch for Linux API
 - defined 29
 - See also* individual functions by name
- FX_closeContext function
 - defined 38
 - using
 - in typical enrollment workflow 22
 - in typical verification workflow 28
- FX_createContext function
 - defined 38
 - using
 - in typical enrollment workflow 18
 - in typical verification workflow 24
- FX_extractFeatures function
 - defined 39
 - using
 - in typical enrollment workflow 20
 - in typical verification workflow 26
- FX_getDisplayImage function
 - defined 42
 - using
 - in typical enrollment workflow 20
 - in typical verification workflow 26
- FX_getFeaturesLen function
 - defined 43
 - using
 - in typical enrollment workflow 18
 - in typical verification workflow 24
- FX_getVersionInfo function, defined 44
- FX_init function
 - defined 45
 - using
 - in typical enrollment workflow 18
 - in typical verification workflow 24
- FX_terminate function
 - defined 46

- using
 - in typical enrollment workflow 22
 - in typical verification workflow 28

G

- getting
 - See retrieving
- getting started with the One Touch for Linux SDK 5

H

- hardware warnings and regulatory information, redistributing 66
- header files
 - data structure definitions extracted from 56
 - function definitions extracted from 29
 - return codes extracted from 60

I

- image
 - See fingerprint image
- important notation, defined 2
- important notice
 - do not specify security level for feature extraction context 52
 - you must be root to run the sample application 5
- initializing
 - comparison module
 - function for 51
 - in typical enrollment workflow 18
 - in typical verification workflow 24
 - device component
 - function for 35
 - in device component workflow 15
 - feature extraction module
 - function for 45
 - in typical enrollment workflow 18
 - in typical verification workflow 24
- introduction to developer guide 1
- italics typeface, uses of 3

K

- kernel mode driver
 - redistributing 65
 - redistributing source files 64
- knowledge-based system 9
 - defined 68

L

- library
 - comparison module
 - redistributing 62

- device component
 - redistributing 62
- feature extraction
 - redistributing 62
- user mode driver
 - redistributing 63

M

- maintenance and use guide for fingerprint reader, redistributing 66
- match 10
 - defined 68
- MC_closeContext function
 - defined 46
 - using
 - in typical enrollment workflow 22
 - in typical verification workflow 28
- MC_createContext function
 - defined 47
 - using
 - in typical enrollment workflow 18
 - in typical verification workflow 24
- MC_generateRegFeatures function
 - defined 47
 - using in typical enrollment workflow 22
- MC_getFeaturesLen function
 - defined 49
 - using in typical enrollment workflow 18
- MC_getSecurityLevel function, defined 50
- MC_getSettings function
 - defined 50
 - using in typical enrollment workflow 18
- MC_getVersionInfo function, defined 51
- MC_init function
 - defined 51
 - using
 - in typical enrollment workflow 18
 - in typical verification workflow 24
- MC_setSecurityLevel function
 - defined 52
 - using in typical verification workflow 24
- MC_SETTINGS data structure, defined 59
- MC_terminate function
 - defined 53
 - using
 - in typical enrollment workflow 22
 - in typical verification workflow 28
- MC_verifyFeaturesEx function
 - defined 53
 - using in typical verification workflow 28

memory

freeing

- allocated by DPFPEnumerateDevices and DPFPGetEvent functions, function for 31
- allocated by DPFPEnumerateDevices function, in device component workflow 15
- allocated by DPFPGetEvent function, in device component workflow 15

N

naming conventions 2

non-match 10

defined 69

notational conventions 2

note notation, defined 2

number of required fingerprint feature sets for fingerprint template creation, retrieving function for 50

in typical enrollment workflow 18

O

One Touch for Linux API

data structures

defined 56

See also individual data structures by name

functions

defined 29

See also individual functions by name

reference 29–59

return codes 60

One Touch for Linux SDK Developer Guide

See developer guide

one-to-one comparison 10

defined 69

online resources 4

overview

of chapters 1

of SDK 9

Q

quick start guide 5

R

redistr directory

licensing requirements for redistribution of files in 62

redistributing the RTE 62

regulatory information, requirement to advise end users of 66

releasing

resources associated with comparison module function for 53

in typical enrollment workflow 22

in typical verification workflow 28

resources associated with feature extraction module function for 46

in typical enrollment workflow 22

in typical verification workflow 28

subscribed fingerprint reader

function for 35

in device component workflow 15

repository 10

resources, additional

See additional resources

resources, online

See online resources

retrieving

events from fingerprint reader

function for 33

in device component workflow 15

information about fingerprint reader, function for 32

number of required fingerprint feature sets for fingerprint template creation

function for 50

in typical enrollment workflow 18

security level of comparison context, function for 50

size of fingerprint feature set

function for 43

in typical enrollment workflow 18

in typical verification workflow 24

size of fingerprint template

function for 49

in typical enrollment workflow 18

software version information

comparison module, function for 51

feature extraction module, function for 44

return codes

device component 60

fingerprint recognition component 60

RTE

redistributing 62

S

sample application

important notice: you must be root to run 5

using 5

SDK

- components of 12
- data flow 12
 - illustrated 13
- security level 12
 - retrieving for comparison context, function for 50
 - setting for comparison context
 - function for 52
 - in typical verification workflow 24
- size of fingerprint feature set, retrieving
 - function for 43
 - in typical enrollment workflow 18
 - in typical verification workflow 24
- size of fingerprint template, retrieving
 - function for 49
 - in typical enrollment workflow 18
- software version information
 - retrieving for comparison module, function for 51
 - retrieving for feature extraction module, function for 44
- subscribing to events from fingerprint reader
 - function for 35
 - in device component workflow 15
- supported DigitalPersona products 4

T

- target audience for developer guide 1
- target false accept rate for comparison context
 - retrieving, function for 50
 - setting
 - function for 52
 - in typical verification workflow 24
- terminating
 - comparison module
 - function for 53
 - in typical enrollment workflow 22
 - in typical verification workflow 28
 - device component
 - function for 36
 - in device component workflow 15
 - feature extraction module
 - function for 46
 - in typical enrollment workflow 22
 - in typical verification workflow 28
- terminology and concepts 9
- token-based system 9
 - defined 69

typefaces, uses of

- bold 3
 - Courier bold 3
 - italics 3
- typographical conventions 3

U

- unsubscribing from events from fingerprint reader
 - function for 37
 - in device component workflow 15
- updates for DigitalPersona software products, URL for downloading 4
- URL
 - DigitalPersona Developer Connection Forum 4
 - Updates for DigitalPersona Software Products 4
- use and maintenance guide for fingerprint reader, redistributing 66
- user mode driver
 - libraries
 - redistributing 63
- using
 - device component API functions 15
 - fingerprint recognition component API functions 16–28
 - sample application 5

V

- verification
 - See fingerprint verification

W

- warning notation, defined 2
- Web site
 - DigitalPersona Developer Connection Forum 4
 - Updates for DigitalPersona Software Products 4