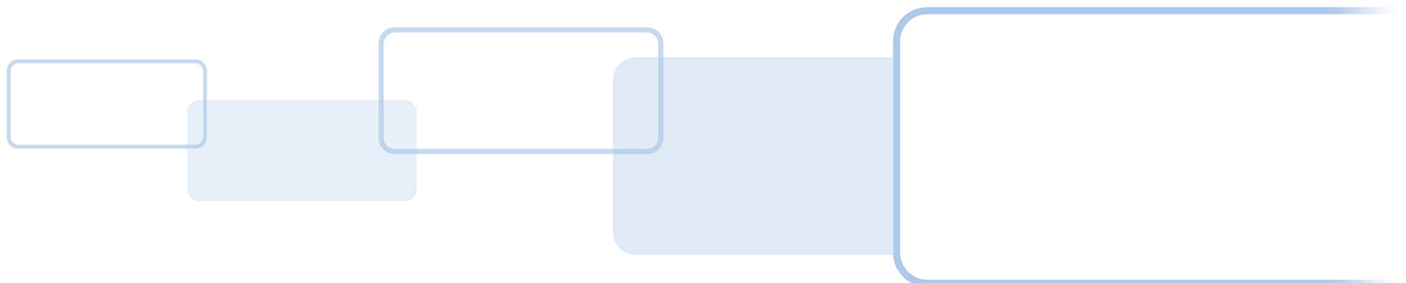


# **DIGITALPERSONA BIOMETRIC SDK**

## **DEVELOPER GUIDE**

January 2020



## Copyright

© 2020 HID Global Corporation/ASSA ABLOY AB. All rights reserved.

This document may not be reproduced, disseminated or republished in any form without the prior written permission of HID Global Corporation.

## Trademarks

DigitalPersona, HID GLOBAL, HID, the HID Brick logo and the Chain Design, are trademarks or registered trademarks of HID Global, ASSA ABLOY AB, or its affiliate(s) in the US and other countries and may not be used without permission. All other trademarks, service marks, and product or service names are trademarks or registered trademarks of their respective owners.

## Revision history

Date	Description	Revision
January 2020	Initial HID release.	A.1

## Contacts

For additional offices around the world, see [www.hidglobal.com/contact/corporate-offices](http://www.hidglobal.com/contact/corporate-offices)

### Americas and Corporate

611 Center Ridge Drive  
Austin, TX 78753  
USA  
Phone: +1 866 607 7339  
Fax: +1 949 732 2120

### Asia Pacific

19/F 625 King's Road  
North Point, Island East  
Hong Kong  
Phone: +852 3160 9833  
Fax: +852 3160 4809

### Europe, Middle East and Africa (EMEA)

Haverhill Business Park Phoenix Road  
Haverhill, Suffolk CB9 7AE  
England  
Phone: +44 (0) 1440 711 822  
Fax: +44 (0) 1440 714 840

### Brazil

Condomínio Business Center  
Av. Ermano Marchetti, 1435  
Galpão A2 - CEP 05038-001  
Lapa - São Paulo / SP  
Brazil  
Phone: +55 11 5514-7100

**HID Global Technical Support:** [www.hidglobal.com/support](http://www.hidglobal.com/support)



# Contents

<b>Section 1: Introduction</b>	<b>9</b>
1.1 Overview	9
1.2 What's new?	9
1.3 How is the DigitalPersona Biometric SDK documentation organized?	10
1.4 Target Audience	10
1.5 How this document is arranged	10
1.6 Data formats and standards	11
<b>Section 2: Background</b>	<b>13</b>
2.1 Biometrics	13
2.2 Features of Biometric Technology	13
2.3 The Basics of Fingerprint Identification	14
2.4 Issues in Fingerprint Recognition Technology	14
<b>Section 3: Developing applications</b>	<b>15</b>
3.1 How Fingerprint Recognition Works	15
3.2 Understanding the Data Flow	15
3.3 Workflow - Enrollment Application	16
3.3.1 Step One - Initialization	17
3.3.2 Step Two - Capture, Extract and Enroll	17
3.3.3 Step Three - Store Data	17
3.3.4 Notes on Enrollment	17
3.4 Workflow - Identifying/Verifying	17
3.4.1 Step One - Initialization	17
3.4.2 Step Two - Capture and Extract	18
3.4.3 Step Three - Identify/Verify	18
3.5 Design Issues for your Application	18
3.5.1 Distributed Processing and Data Flow	18
3.5.2 Data Compression	19
3.5.2.1 WSQ Compression	19
3.5.2.2 PNG Compression	19
3.5.3 NIST Fingerprint Image Quality (NFIQ)	19
3.5.4 Determining an Acceptable Level of Error	19

3.5.4.1	Setting the Error Threshold when Identifying a Fingerprint in a Collection. . . . .	19
3.6	Defining the Data Retention Policy. . . . .	20
3.6.1	Specifying the Fingers to Be Scanned. . . . .	20
3.6.2	Optimizing Fingerprint Applications . . . . .	21
<b>Section 4:</b>	<b>Working with fingerprint readers . . . . .</b>	<b>23</b>
<b>Section 5:</b>	<b>Upgrading from previous SDKs . . . . .</b>	<b>25</b>
5.1	Working with Existing Data Created with Previous SDKs. . . . .	25
5.2	Exchanging Data with Applications Using Previous SDKs. . . . .	25
5.2.1	Upgrading Applications from Previous SDKs. . . . .	25
5.3	Converting Applications from the One Touch SDK . . . . .	26
5.4	Working with Data Created with Gold and One Touch SDKs. . . . .	26
5.5	Changes in Terminology from Previous Usage (Gold and One Touch) . . . . .	26
5.6	Upgrading from Digital Persona UPOS for OPOS/JavaPOS. . . . .	27
<b>Section 6:</b>	<b>Using the SDK . . . . .</b>	<b>29</b>
6.1	What's In the SDK?. . . . .	29
6.2	FingerJet Engine . . . . .	30
<b>Section 7:</b>	<b>The C++ APIs . . . . .</b>	<b>31</b>
7.1	DP Capture API. . . . .	31
7.1.1	Library Management . . . . .	31
7.1.2	Fingerprint Capture Device Management. . . . .	31
7.1.3	Capturing Fingerprints . . . . .	32
7.2	FingerJet Engine API. . . . .	32
7.2.1	Library Management . . . . .	33
7.2.2	Select Fingerprint Matching Engine . . . . .	33
7.2.3	Extract FMD. . . . .	33
7.2.4	Identify Fingerprint . . . . .	33
7.2.5	Enrollment . . . . .	33
7.2.6	Format conversion. . . . .	34
7.2.7	Advanced Diagnostics. . . . .	34
7.2.8	Wavelet Scalar Quantization (WSQ) Compression. . . . .	35
7.2.9	Raw WSQ Compression . . . . .	35
7.2.10	NIST Fingerprint Image Quality (NFIQ). . . . .	36
<b>Section 8:</b>	<b>The Java API . . . . .</b>	<b>37</b>
8.1	Importing the Digital Persona Java package . . . . .	37
8.2	Getting Detailed Documentation . . . . .	37
8.3	Using the Package . . . . .	37

8.3.1	Main Access Point .....	37
8.3.2	UareUException .....	37
8.3.3	Getting a List of Available Readers .....	38
8.4	Working with Readers.....	38
8.4.1	Capturing Fingerprints .....	38
8.4.2	Streaming Fingerprints .....	39
8.5	Accessing the FingerJet Engine .....	39
8.5.1	Creating FMDs from images.....	39
8.5.2	Identification and Comparison.....	40
8.5.2.1	Identify().....	40
8.5.2.2	Compare() .....	40
8.5.3	Enrollment .....	41
<b>Section 9:</b>	<b>The Javascript API .....</b>	<b>43</b>
9.1	Overview .....	43
9.2	Using the fingerprint library.....	43
9.3	Fingerprint.WebApi methods .....	44
9.3.1	enumerateDevices().....	44
9.3.2	getDeviceInfo().....	44
9.3.3	DeviceInfo .....	45
9.3.4	StartAcquisition().....	46
9.3.5	stopAcquisition.....	47
9.4	Fingerprint.WebApi events .....	47
9.4.1	DeviceConnected.....	48
9.4.2	DeviceDisconnected .....	48
9.4.3	SamplesAcquired .....	48
9.4.4	QualityReported.....	48
9.4.5	ErrorOccured.....	48
9.4.6	CommunicationFailed .....	49
9.5	Enumerations .....	49
9.5.1	DeviceUidType .....	49
9.5.2	DeviceModality .....	49
9.5.3	DeviceTechnology .....	49
9.5.4	SampleFormat.....	49
9.5.5	QualityCode.....	50
9.6	Sample Format details .....	50
9.6.1	Raw .....	50
9.6.2	Intermediate .....	51

9.6.3	Compressed (WSQ).....	52
9.6.4	PNG.....	53
<b>Section 10:</b>	<b>Integration with web applications .....</b>	<b>55</b>
10.1	RAW .....	55
10.1.1	ANSI or ISO minutiae standards are mandated .....	56
10.1.2	Images are kept or sent to a third party .....	56
10.2	INTERMEDIATE.....	57
10.2.1	Biometric enrollment.....	57
10.2.1.1	User verification .....	57
10.3	PNG.....	58
10.4	WSQ .....	58
<b>Section 11:</b>	<b>The .NET API .....</b>	<b>61</b>
11.1	Importing the Digital Persona .NET package .....	61
11.2	Getting Detailed Documentation .....	61
11.3	Using the Package .....	61
11.3.1	Main Access Points.....	61
11.3.2	SDKException .....	62
11.3.3	Serialization .....	62
11.3.4	IEnumerables in the .NET Wrapper.....	62
11.4	Working with Readers.....	63
11.4.1	Capturing Fingerprints .....	65
11.4.2	Streaming Fingerprints .....	65
11.5	Managing Fingerprint Data.....	66
11.6	Analyzing and Managing Fingerprints (FingerJet Engine) .....	67
11.6.1	Creating FMDs from images.....	68
11.6.2	Comparing Fingerprints .....	69
11.7	Enrollment .....	70
11.7.1	Common Data Structures for Results.....	71
11.7.2	Pre-Built Controls for Enrollment and Identification.....	72
<b>Section 12:</b>	<b>The ActiveX API .....</b>	<b>73</b>
12.1	Importing the Digital Persona ActiveX package .....	73
12.2	Getting Detailed Documentation .....	73
12.3	Using the Package .....	73
12.3.1	Main Access Points.....	73
12.3.2	SDKException .....	74
12.3.3	Serialization .....	74

12.4	Working with Readers.....	74
12.4.1	A Note About Internet Explorer and Process Merging.....	74
12.4.2	Class Diagrams.....	75
12.5	Managing Fingerprint Data.....	77
12.6	Accessing the FingerJet Engine.....	78
12.6.1	Creating FMDs from images.....	79
12.6.2	Identification and Comparison.....	79
12.6.3	Enrollment.....	80
12.6.4	Common Data Structures for Results.....	80
12.6.5	Pre-Built Controls for Enrollment and Identification.....	81
<b>Section 13:</b>	<b>The JavaPOS API.....</b>	<b>83</b>
13.1	Terminology Note.....	83
13.2	Working with Fingerprint Data in JavaPOS.....	83
13.2.1	Fingerprint Data for Raw Images (Captures).....	84
13.2.2	Fingerprint Data for Captures and Enrollment Templates (BIRs).....	84
13.2.3	Working with Digital Persona Record Formats.....	84
13.2.4	Converting to a Different Data Format.....	85
13.3	Getting Device-Specific Information with DirectIOEvent.....	85
13.4	Implementation Notes.....	86
13.5	Exceptions.....	88
13.6	Device-Related Error Codes.....	89
<b>Section 14:</b>	<b>The OPOS API.....</b>	<b>91</b>
14.1	Terminology Note.....	91
14.2	Working with Fingerprint Data in OPOS.....	91
14.2.1	Fingerprint Data for Raw Images (Captures).....	92
14.2.2	Fingerprint Data for Captures and Enrollment Templates (BIRs).....	92
14.2.3	Working with Digital Persona Record Formats.....	92
14.2.4	Converting to a Different Data Format.....	93
14.3	Getting Device-Specific Information with DirectIOEvent.....	93
14.3.0.1	DataEvent.....	93
14.3.0.2	DirectIOEvent.....	93
14.3.0.3	StatusUpdateEvent.....	95
14.4	Implementation Notes.....	95
14.5	Exceptions.....	99
14.6	Device-Related Error Codes.....	99

**Section 15: The Objective-C API . . . . . 101**

15.1 DP Capture API . . . . . 101

15.1.1 Fingerprint Capture Device Management . . . . . 101

15.1.2 Capturing Fingerprints . . . . . 101

15.1.3 ReaderDelegate Methods . . . . . 102

15.1.4 Reader Notifications . . . . . 102

15.1.5 Streaming Fingerprints . . . . . 102

**Section 16: Application Notes . . . . . 103**

16.1 General Fingerprint Issues . . . . . 103

16.2 Minex certification . . . . . 103

16.3 C/C++ Issues . . . . . 103

**Section 17: Glossary . . . . . 105**

17.1 General Terms . . . . . 105

17.2 Fingerprint Data . . . . . 105

17.3 Fingerprint Devices . . . . . 106

17.4 Fingerprint Recognition Terms . . . . . 106

17.5 Recognition Accuracy . . . . . 107



# Chapter 1

## 1 Introduction

This chapter provides an overview of the functionality and features of the DigitalPersona Biometric SDK.

### 1.1 Overview

The DigitalPersona Biometric SDK allows you to add fingerprint capture and recognition to applications developed for a wide-variety of platforms. Significant features of the DigitalPersona Biometric SDK include:

- Performing 1-to-many fingerprint identification automatically
- Support for multiple data formats, including Digital Persona data formats as well as ANSI and ISO fingerprint images and minutiae data
- FingerJet Engine that meets the PIV performance thresholds for fingerprint minutiae data generation required by NIST.

The SDK provides support for developing applications on the following platforms using the languages/frameworks shown below (X indicates a supported combination).

API	Windows	Linux	Android	iOS
C/C++	X	X		X
Java	X	X	X	
.NET	X			
ActiveX	X			
JavaPOS	X	X		
Javascript	X			
Objective-C				X

The DigitalPersona Biometric SDK supports:

- Development on Windows for target devices based on Windows
- Development on Linux for target devices based on Linux or Android
- Development on MacOS for target devices based on iOS

### 1.2 What's new?

New features in version 3.3

- Added support for iOS. Note that Matching engine features are not supported on iOS.

New features in version 3.1

- Added support for Nomad 30/Pocket 30 family of fingerprint readers and modules.

- Added support for arm64-v8a and Intel x86 platforms.

## 1.3 How is the DigitalPersona Biometric SDK documentation organized?

The Digital Persona documentation set consists of a family of books:

- The DigitalPersona Biometric SDK Developer's Guide (this manual) describes the architecture and organization of the SDK plus an overview of language/framework support.
- Platform Guides for Windows, Linux, Android and iOS provide details about installation, requirements and code samples for devices based on the respective operating systems.

In addition, Javadoc and Doxygen documentation provides details of the methods, parameters and data structures for the .NET, ActiveX and Java APIs.

## 1.4 Target Audience

This manual is aimed at developers who already have a working knowledge of their development environment and their chosen reader platform. We make the following assumptions.

- C/C++ API - We assume you have a working knowledge of the C++ language.
- Objective-C API - We assume you have a working knowledge of the Objective-C language.
- .NET - We assume that you know how to develop for .NET and understand .NET concepts like static classes, IDisposable and IEnumerable.
- ActiveX - We assume that you know how to develop with ActiveX.
- Java - We assume a working knowledge of the Java language.
- JavaScript - We assume a working knowledge of Javascript.
- OPOS and JavaPOS - We assume knowledge of the UPOS (OPOS and JavaPOS) specification version 1.13 and familiarity with Point of Sale applications and developing with OPOS/JavaPOS.

## 1.5 How this document is arranged

[Introduction](#) (this chapter), provides an overview of the features and standards compliance of the DigitalPersona Biometric SDK.

[Background](#), contains a brief introduction to fingerprint recognition and fingerprint biometrics.

[Developing applications](#), describes the context and issues for developing applications using the DigitalPersona SDK. This chapter shows how data flows among the Digital Persona Biometric SDK components, and typical workflows.

[Working with fingerprint readers](#), contains information about working with fingerprint reader hardware.

[Upgrading from previous SDKs](#), discusses how to upgrade your application to use the current API functions and data formats, as well as a mapping between the One Touch terminology and the terminology used in the DigitalPersona Biometric SDK.

[Using the SDK](#), provides an overview of the SDK and describes how to generate API documentation using Doxygen.

[The C++ APIs](#), provides an overview of the functions in the Objective-C libraries.

[The Java API](#), describes the interfaces for the Java libraries.

[The Javascript API](#), describes the interfaces for the JavaScript Fingerprint library.

[The .NET API](#), describes the classes and methods in the .NET libraries as well as the .NET controls.

[The ActiveX API](#) describes the interfaces for the ActiveX libraries and controls.

[The JavaPOS API](#), describes the JavaPOS API including data management, implementation notes, error codes and exceptions.

[The OPOS API](#), describes the OPOS API including data management, implementation notes, error codes and exceptions.

[The Objective-C API](#), provides an overview of the functions in the C libraries.

[Application Notes](#) provides additional suggestions for getting your application to work.

[Glossary](#), lists the terminology specific to fingerprint recognition applications and to the DigitalPersona Biometric SDK.

## 1.6 Data formats and standards

The DigitalPersona Biometric SDK supports three data types:

- Digital Persona
- ANSI
- ISO





# Chapter 2

## 2 Background

---

In this chapter, we discuss the basics of fingerprint recognition. This chapter is not intended to be exhaustive, rather we're going to give you enough background knowledge to develop your own application more effectively.

For a more detailed overview, we recommend the Handbook of Fingerprint Recognition by D. Maltoni, M. Maio, A. Jain, and S. Prabhakar, published by Springer, 2nd edition, 2009.

### 2.1 Biometrics

Identifying individuals based on their distinctive anatomical (fingerprint, face, iris, hand geometry) and behavioral (signature, voice) characteristics is called biometrics. Because biometric identifiers cannot be shared or misplaced, they intrinsically represent an individual's identity. Biometrics is quickly becoming an essential component of effective identification solutions. Recognition of a person by their body, then linking that body to an externally established "identity", forms a powerful authentication tool.

Biometric identification helps to reduce fraud, and enhance user convenience. Among the different biometric identification methods, fingerprint recognition technology has a good balance of qualities including accuracy, throughput, size and cost of readers, maturity of technology and convenience of use, making it the dominant biometric technology in commercial applications. This chapter provides an overview of the functionality and features of the DigitalPersona Biometric SDK.

### 2.2 Features of Biometric Technology

Biometric solutions offer many advantages that other technologies cannot provide.

*Uniqueness* - Fingerprints from each one of our ten fingers are distinctive, different from one another and from those of other persons. Even identical twins have different fingerprints.

*Convenience* - Users no longer have to remember multiple, long and complex, frequently changing passwords or carry multiple keys.

*Non-repudiation* - Ensures the user is present at the point and time of recognition and later cannot deny having accessed the system.

*Non-transferable* - Cannot be shared, lost, stolen, copied, distributed or forgotten unlike passwords, PINs, and smart cards.

*Proven* - Long history of successful use in identification tasks - the U.S. and other countries have extensive real-world experience with fingerprint recognition. Fingerprints have been used in forensics for well over a century and there is a substantial body of scientific studies and real world data supporting the distinctiveness and permanence of fingerprints.

## 2.3 The Basics of Fingerprint Identification

The skin on the inside surfaces of our hands, fingers, feet, and toes is “ridged” or covered with concentric raised patterns. These ridges are called friction ridges and they provide friction making it easier for us to grasp and hold onto objects and surfaces without slippage. The many differences in the way friction ridges are patterned, broken, and forked make ridged skin areas, including fingerprints, distinctive.

The distinctiveness of fingerprints is well established. The underlying biological persistence of fingerprint characteristics is also a well established fact reported in various fingerprint studies conducted in different scientific fields over the past century.

## 2.4 Issues in Fingerprint Recognition Technology

In a perfect world, it would be a simple matter to determine whether two fingerprints were from the same finger-- the images would be identical or they would not. However, even though our fingerprints do not change over time, the fingerprint images can vary a lot, especially for some people. For example, certain skin conditions and wear due to manual labor can affect fingerprint images. This makes fingerprint recognition a very challenging problem that does not have a perfect solution. As a result, captured fingerprint images are often not a perfect match to the stored image from the same finger.

Fingerprint images from two different fingers of two different people can look similar, especially when, because of worn fingerprints or temporary creases, there is very little information left about the actual fingerprint. The larger the population you are working with, the more likelihood of similar fingerprint images.

Fingerprint recognition software needs to address these issues. We'll discuss that more in later sections of this guide.



# Chapter 3

## 3 Developing applications

---

This chapter covers the following topics.

- How fingerprint recognition works
- Data structures and data flows among components
- Typical workflows
- Design issues and tradeoffs.

The JavaPOS and OPOS APIs conform to the UPOS specification and therefore do not have exactly the same data structures and workflow as the other APIs in the DigitalPersona Biometric SDK. See [Section 13 The JavaPOS API](#) and [Section 14 The OPOS API](#) for details.

Note: Fingerprint matching and compression are not supported on iOS.

### 3.1 How Fingerprint Recognition Works

Fingerprint recognition works in two stages:

- First, users are enrolled with the system--their fingerprints are captured and stored in a database.
- Next, when a person needs to be given access (e.g., to open a door or to log in to a computer), they simply scan their finger on the fingerprint reader.

In terms of application development, this typically requires the developer to build the following components:

1. An application that enrolls users:
  - Captures multiple fingerprints for at least two fingers from a fingerprint reader.
  - Checks image quality to ensure that a good quality scan is obtained.
  - Extracts the fingerprint minutiae.
  - Saves the fingerprint images and/or minutiae in a database.
2. A service(s)/application(s) that identifies/verifies people:
  - Captures a fingerprint from a fingerprint reader.
  - Extracts the fingerprint minutiae.
  - Compares fingerprint with enrolled fingerprints to identify a user from a list or verify a specific user.

This SDK provides fingerprint capture, extraction, enrollment and identification/verification functions to help you develop these components.

### 3.2 Understanding the Data Flow

When building a fingerprint recognition application, the data flow consists of:

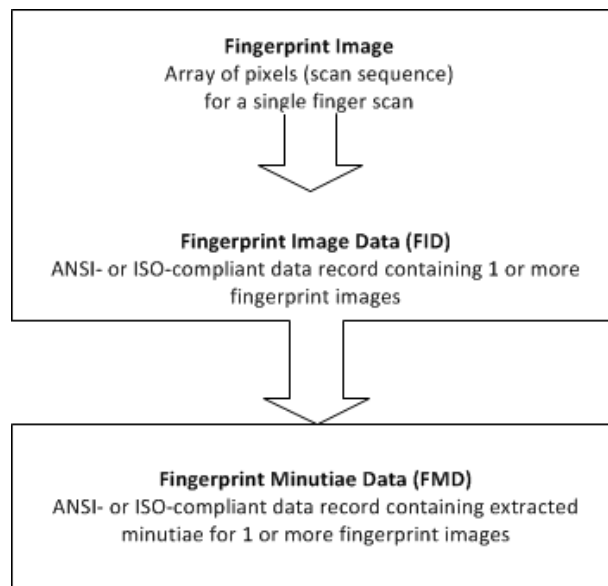
1. Capture a Fingerprint Image (scan) from the fingerprint reader. The resulting Fingerprint Image Data (FID) contains one or more fingerprint images, called a Fingerprint Image Views (FIVs). A typical FID for fingerprint recognition applications contains only one FIV but we also support multiple views (e.g., if there are multiple fingers from one individual or multiple images from a single finger stored in a single FID).

Each FIV (fingerprint) is approximately 140K in size.

2. Extract the fingerprint features. During extraction, Fingerprint Minutiae Data (FMD) is created, with each fingerprint stored in a Fingerprint Minutiae View (FMV) in the FMD. An FMV in an FMD takes no more than 1.5K (maybe less depending on the fingerprint). FMDs are used for identifying users in a collection and verifying specific users.

The ANSI and ISO standards permit multiple views but the DigitalPersona Biometric SDK creates only single-view FIDs and FMDs.

This data flow is shown below.



A terminology note: in the past, enrollment fingerprints were stored as templates and fingerprints to be identified/verified were created as feature sets. This data model is still supported when using the Digital Persona data format - that is, there are two kinds of FMD, depending on whether the data is for an enrolled fingerprint or for a fingerprint to be identified/verified. When using ANSI and ISO standard data, the template/feature set terminology does not apply since the ANSI/ISO standards do not make that distinction. The data flow for legacy SDK data may be different, as described in Working with Data Created with Gold and One Touch SDKs on page 20.

### 3.3 Workflow - Enrollment Application

During the enrollment process, one or more fingers are scanned for each person. We recommend that you enroll at least two fingers (more is recommended) because in the event of an accident or injury to one finger, another enrolled finger can be used to identify the individual.

The enrollment application needs to perform the following steps to enroll a single finger from a user:



### 3.3.1 Step One - Initialization

Initialize the library. Discover the available readers and open a connection to the reader.

### 3.3.2 Step Two - Capture, Extract and Enroll

1. Begin the enrollment process.
2. Capture a series of fingerprint scan(s); for each scan,
  - Create an FID,
  - Extract fingerprint minutiae and create the FMD,
  - Add the FMD to the pool of FMDs for enrollment.
3. Continue to capture fingerprints until the enrollment process has enough FMDs to complete the enrollment. (The enrollment functions evaluate the FMDs and select the best image -- typically several scans are required.)
4. Create the enrollment FMD and release resources.

### 3.3.3 Step Three - Store Data

Store the enrollment FMD. Many applications keep only the enrollment FMD because of space constraints or policy decisions. You cannot use FIDs for identification, so even if you choose to keep the FIDs, you must also store the FMD for each individual.

### 3.3.4 Notes on Enrollment

Before storing, you may want to check for existing entries that match the new entry -- applications like law enforcement, banking or voting registration, may not allow duplicate enrollments.

The capture/extract minutiae part of the enrollment process is the same as for capturing/extracting minutiae for the purpose of verifying/identifying users. If you wish, you can enroll users without using the enrollment functions (by simply capturing, extracting minutiae and storing the resulting FMD). However we recommend that you use the enrollment functions to create the best quality enrollment FMDs.

The enrollment process is slightly different in each API. Consult the chapters that describe the various APIs to determine the specifics for your language. For JavaPOS and OPOS, the enrollment process is described in the specification.

## 3.4 Workflow - Identifying/Verifying

Fingerprint recognition involves two types of operation:

*Identification* - Comparing a fingerprint against the database of enrolled fingerprints and confirming that the fingerprint is enrolled (e.g., to open a door there may be many authorized users).

*Verification* - Comparing a fingerprint against a specific user's enrolled fingerprint(s) to verify a specific person's identity (e.g., when the user types their name and then uses a fingerprint rather than a password).

To perform these operations, your application needs to do the following steps:

### 3.4.1 Step One - Initialization

Initialize the library. Discover the available readers and open a connection to a reader.

### 3.4.2 Step Two - Capture and Extract

Wait for a fingerprint. When a fingerprint is detected, capture the image and create an FID.

Extract fingerprint minutiae and create an FMD.

This sequence is exactly the same as for the capture/extraction process during enrollment.

### 3.4.3 Step Three - Identify/Verify

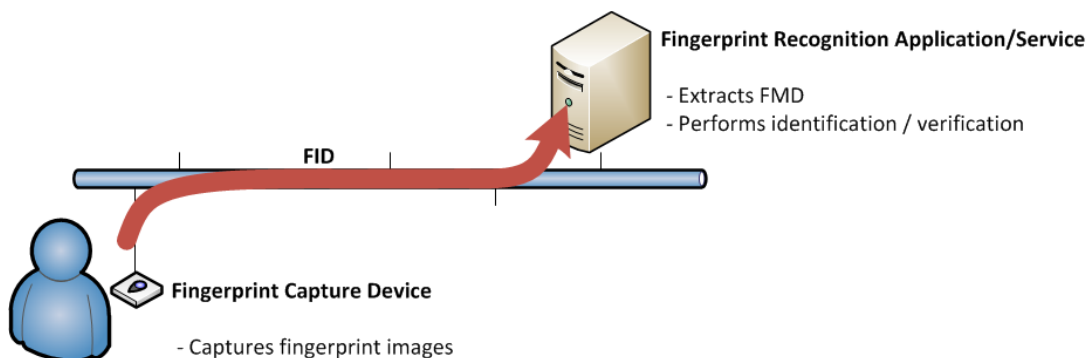
Call the appropriate function to verify a specific person OR to identify a valid user.

## 3.5 Design Issues for your Application

### 3.5.1 Distributed Processing and Data Flow

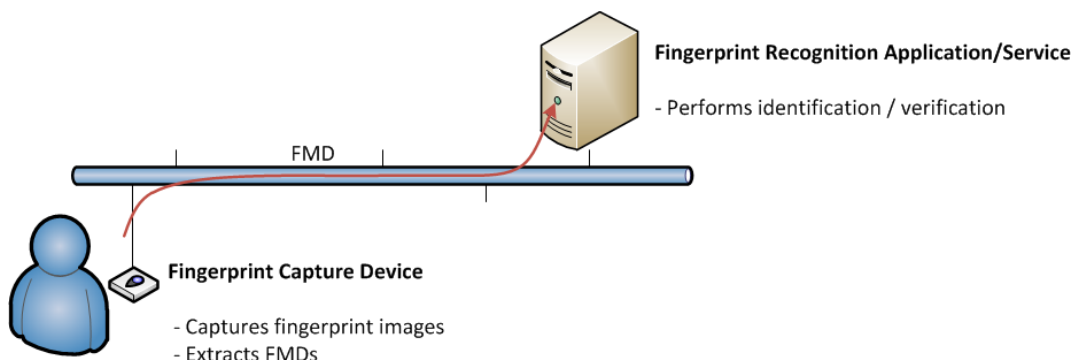
Depending on the capabilities of your fingerprint reader, you can capture FIDs and send them to another machine for processing OR the fingerprint reader can extract the FMD and transmit only the much smaller FMD files. Thus the application can be designed in these two ways:

1. The fingerprint capture device can simply capture a fingerprint image and transmit the image to a server for processing as shown in the image below. Since FIDs are large (around 100K - 140K), this means that you need a faster connection, but there is less computing power required by the fingerprint reader.



Optionally, the fingerprint capture device can compress the image before transmitting it.

2. Another alternative is to develop software for the fingerprint capture device to capture the fingerprint image AND extract the fingerprint features to create an FMD. The FMD is then transmitted to the server for processing, as shown below. FMDs are 1.5K or less and so require less bandwidth and speed.



### 3.5.2 Data Compression

If you use compression to save bandwidth or storage space, there are two options:

The DigitalPersona Biometric SDK provides compression using Wavelet Scalar Quantization (WSQ). WSQ is a wavelet-based compression standard developed by NIST specifically for fingerprint data.

You can compress the data yourself using a lossless compression such as PNG.

#### 3.5.2.1 WSQ Compression

WSQ compression allows compression of 15:1 to 12:1. For more details, see [Section 7.2.8 Wavelet Scalar Quantization \(WSQ\) Compression](#).

#### 3.5.2.2 PNG Compression

If you are using PNG as a compression algorithm. The DigitalPersona Biometric SDK does not support PNG compression directly, so you would have to capture the fingerprint as a raw image and compress it to PNG. After transmission, you must convert the PNG file back into a raw image for minutiae extraction and identification/verification.

### 3.5.3 NIST Fingerprint Image Quality (NFIQ)

The NFIQ score indicates the quality of a fingerprint sample. The DigitalPersona Biometric SDK provides NFIQ calculation using code developed by NIST. The NFIQ score is in the range 1 – 5, with 1 being the best and 5 being not suitable for feature extraction. For more details, see [Section 7.2.10 NIST Fingerprint Image Quality \(NFIQ\)](#).

### 3.5.4 Determining an Acceptable Level of Error

When identifying fingerprints, you want to identify strictly enough that you do not let unauthorized people have access (false positives) but also do not inconvenience legitimate users by rejecting their fingerprints (false negatives). Note that some people will always experience more false rejections -- the rate of false negatives is a statistical measure, but individuals may experience higher rejection rates, based on their specific fingerprint characteristics.

There is a trade-off between the frequencies of false positive and false negative errors. Applications have control over this trade-off by specifying the threshold for the required degree of similarity between two fingerprint images in order to call it a match.

When choosing the identification threshold, note that increasing the false positive error rate by a factor of 100 will reduce the false negative error rate only approximately by a factor of 2. There will always be some false negatives. As the result, every practical fingerprint recognition system should have an alternative means to establish and prove identity, without using fingerprints.

#### 3.5.4.1 Setting the Error Threshold when Identifying a Fingerprint in a Collection

When a fingerprint is scanned, the first step is to identify the fingerprint against a set of stored FMDs.

- If you are trying to confirm that a user is allowed access, you will want to identify the fingerprint against all the valid FMDs that you have stored.
- If you are trying to confirm the identity of a specific person, you must identify against all FMDs for that individual (typically at least two fingers are stored for each user).

The identification function compares an FMV against a collection of FMDs to produce the candidate list. You can specify the maximum desired number of candidates: a smaller number can make the execution faster. The candidate list is sorted by the dissimilarity score, the lower the score the closer the candidate to the beginning of the list. The best match is the first candidate in the list.

Your threshold determines the trade-off between false positive and false negative error rates where:

- 0 = no false positives
- maxint (#7FFFFFFF or 2147483647) = fingerprints do not match at all
- Values close to 0 allow very few false positives; values closer to maxint allow very poor matches (a lot of false positives) in the candidate list. The table below shows the relationship between the threshold values and the false positive identification error rates observed in our test. Note: the actual false positive identification error rates in your deployment may vary.

Your threshold	Corresponding False Positive Identification Rate	Expected number of False Positive Identifications	Numeric Value of Threshold
.001 * maxint	.1%	1 in 1,000	2147483
.0001 * maxint	.01%	1 in 10,000	214748
.00001 * maxint	.001%	1 in 100,000	21474
1.0e-6 * maxint	.0001%	1 in 1,000,000	2147

For many applications, a good starting point for testing is a threshold of 1 in 100,000. If you want to be conservative, then you will want to set the threshold lower than the desired error rate (e.g., if you want an error rate that does not exceed 1 in 100,000, you might set the threshold to 1 in 1,000,000).

## 3.6 Defining the Data Retention Policy

After a fingerprint scan, an FID is created, which contains the actual image. Each fingerprint image takes roughly 140K of storage. With modern computers, that is not a huge amount, but each enrolled user may have several fingerprints scanned. Multiplied by the number of potential users, this can add up to a fair amount of data.

To identify fingerprints, you must extract the fingerprint characteristics to create an FMD, which is < 1.5K per fingerprint.

Some applications choose to retain the full image records, but other companies discard the image record and retain only the minutiae data in the form of an FMD.

**IMPORTANT:** If you discard the image record, you cannot reconstruct the original fingerprint image from the FMD, the FMD is only useful for identifying/verifying fingerprints.

From time to time Crossmatch may release a new version of the DigitalPersona Biometric SDK that will provide improved accuracy in the feature extraction process. If you do not retain the fingerprint images, you will not be able to redo the feature extraction using the new version of the SDK/runtime in order to take advantage of these improvements.

### 3.6.1 Specifying the Fingers to Be Scanned

When you enroll people into your system, you will usually want more than one finger enrolled. This allows for injury and makes it easier for people who have fingerprints that are difficult to recognize. For some applications you may want to scan all ten fingers or take multiple fingerprint impressions for individual fingers.

A typical policy would be to require both index fingers or both thumbs to be scanned. Thumbs are typically the worst in terms of recognition accuracy and thumbs require different capture devices with larger area and different ergonomics. Where possible, avoid any industrial design forcing users to use thumbs.

The preferred approach is to enroll, at a minimum, both index and both middle fingers. Middle fingers are usually the best, probably because people have almost as good dexterity with middle fingers as with index fingers, and yet middle fingers have fingerprints that are typically less worn than index fingers.

Some solutions are ergonomically designed in a way that only the right or only the left hand can be used conveniently. In this case the right hand is better (probably because the majority of people are right handed), and at least three fingers need to be enrolled: index, middle and ring fingers.

Ergonomics and the correct finger placement are extremely important. Poor ergonomics can easily increase the false negative identification rate by a factor of 5 to 10. The system users need to be aware of correct finger placement for best results.

### 3.6.2 Optimizing Fingerprint Applications

To identify a fingerprint against a large database can take a considerable amount of time and create unacceptable delays between the fingerprint scan and the user authorization. Identifying fingerprints will be faster if the database of fingerprints is in memory rather than retrieved from disk. If you have a large database of users, you may need to provision your server with an appropriate amount of RAM to handle the searches.

This SDK is not optimized for large scale identification. If you are developing such an application, you may want to contact Crossmatch to get help selecting the technology that will best fit your needs.

To ensure the fastest response time, you must weigh whether it will be faster to transmit the image record to a server for FMD extraction or whether it is faster to extract the FMD on the reader and transmit only the FMD to the server for identification/verification.

For readers that are used by a limited number of people (e.g., kiosks or pharmacy cabinets), you may have the device identify fingerprints against a limited set of FMDs. However this requires that you keep the FMDs in the device in sync with your central database, to ensure that new employees are able to gain access and departing employees' privileges are revoked quickly.





# Chapter 4

## 4 Working with fingerprint readers

---

The DigitalPersona Biometric SDK works with the following fingerprint readers:

- DigitalPersona 4500 Fingerprint Reader Rev. 103
- DigitalPersona 5xxx Fingerprint Readers (5100, 5160, 5200 and 5300)
- Eikon 510 and 710 Fingerprint Readers
- Nomad 30 and Nomad 30 Pocket Fingerprint Readers
- Lumidigm M21x modules and readers

However, please note the following limitations.

- The DigitalPersona 4xxx and Eikon readers do not support image streaming.
- The DigitalPersona 4xxx and Eikon readers do not support selection of alternate image processing formats.
- For the DigitalPersona 5xxx, Nomad 30 and Nomad 30 Pocket Readers, only two readers may be used simultaneously. You can use two of the same model readers or two different models, but they must be connected to separate USB controllers.

Fingerprint readers can lose their calibration as a result of changes in temperature, humidity and ambient light. Humidity is the most important environmental factor affecting calibration. DigitalPersona fingerprint readers are self-calibrating, but you still might want to set up your application to check periodically that no additional calibration is needed.

If the fingerprint reader is not giving clear readings:

- Try cleaning it:
  - For the 4xxx readers, clean the gel surface with sticky tape. Gently dab it with the sticky side of the tape. Do not rub it with paper and do not get it wet.
  - For the 5xxx readers, in addition to sticky tape, you can use a damp wipe. Do not clean with compressed air and do not use industrial cleaners or solvents.
- Make sure that you are touching the fingerprint reader with the pad of your finger, not the tip. The most details (minutiae) occur roughly midway between the first joint and the tip.
- If your fingers are very dry, try touching your forehead with the pad of the finger you are trying to scan and then rescanning your fingerprint.

Good ergonomics in the mounting of your fingerprint reader can significantly improve the quality of fingerprint scans. Be sure that the reader is mounted in a way that is convenient for users to touch properly, such that the large area of the pad of the finger is captured. The angle at which the reader is mounted as well as the rim around the sensing area can make it difficult for people with large fingers or long fingernails to have proper contact between the pad of the finger and the sensing area.

If your fingerprint reader becomes non-responsive from an electrostatic shock you may need to perform a hardware reset.



Your application should check the reader status between fingerprint scans to ensure that the hardware has not experienced an error condition.





# Chapter 5

## 5 Upgrading from previous SDKs

---

### 5.1 Working with Existing Data Created with Previous SDKs

The DigitalPersona Biometric SDK supports Digital Persona, ANSI, and ISO formats. You may choose your format according to your application requirements.

However, the data formats are not interoperable and only one data format can be used in a single application.

For example, when passing an array of fingerprint templates into the identification function, all the templates in the array must be in the same format. If your application has existing data in a specific format, you must continue to use that format for all of your data.

**IMPORTANT:** It is not possible to convert data from one format to another. If you decide to move from one format to another, you must re-enroll your user population.

### 5.2 Exchanging Data with Applications Using Previous SDKs

Applications written with the DigitalPersona Biometric SDK APIs are able to exchange data with applications developed using previous Digital Persona or One Touch products. The table below shows which Digital Persona APIs may be used to develop applications that will exchange data with existing applications:

Previous SDK used by existing application	Compatible Digital Persona APIs
Gold SDK/Fingerprint Recognition Software	C/C++, Java, ActiveX
One Touch SDKs	C/C++, Java, ActiveX
Digital Persona UPOS for JavaPOS	JavaPOS
Digital Persona UPOS for OPOS	OPOS

The DigitalPersona Biometric SDK does not support the custom encryption keys that are supported by the Gold and One Touch SDKs.

#### 5.2.1 Upgrading Applications from Previous SDKs

To convert applications developed using Gold and One Touch SDKs to use the DigitalPersona Biometric SDK requires software modifications to use the Digital Persona APIs. The Digital Persona APIs are described in detail in the other chapters of this manual.

Applications developed with Digital Persona UPOS for JavaPOS or UPOS for OPOS can be upgraded to use the JavaPOS and OPOS APIs of the DigitalPersona Biometric SDK, since both the previous and the current product support the UPOS specification. More details are provided below in [Section 5.6 Upgrading from Digital Persona UPOS for OPOS/JavaPOS](#).

## 5.3 Converting Applications from the One Touch SDK

To convert an application from the One Touch SDK, install the DigitalPersona Biometric SDK as described above and use this documentation to modify your applications to use the new API.

Be sure to install the DigitalPersona Biometric SDK in a new folder on your development machine so that your existing files do not get overwritten.

Note that when you install the DigitalPersona Biometric SDK on the target machine, the One Touch drivers will be overwritten with new drivers that are compatible with both your existing applications and new applications based on the DigitalPersona Biometric SDK.

The files installed on the target machine include both drivers and SDK files. If you retain the One Touch SDK files on the device, you will need up to an additional 120K for the new DigitalPersona Biometric SDK files. If you retain the old SDK files on the device, you can run applications based on either the old or the new SDK. The two SDKs can coexist and your existing applications can run using the older run-time environment while you update your programs or develop new applications based on the new SDK. Applications based on the old and new SDKs can run on the same hardware, but not simultaneously.

Note that the data from existing applications based on the One Touch SDK is fully compatible with applications based on the DigitalPersona Biometric SDK.

## 5.4 Working with Data Created with Gold and One Touch SDKs

The Digital Persona Gold SDK and One Touch SDK products used a different format for minutiae data (which were called feature sets). In the Gold and One Touch formats, there were three different data types for minutiae data which reflected the intended use of the data:

- *Pre-registration features* (Gold SDK), or *Pre-registration Feature Set* to be used for Enrollment (OneTouch SDK); Pre-registration features were intended only for creation of Registration features. An application needed to collect four fingerprints of the finger to enroll, extract pre-registration features, and pass it to the SDK to produce registration features.
- *Registration features* (Gold SDK), or *Fingerprint Template* (OneTouch SDK); *Registration features* were intended to be stored in a database as an enrolled finger.
- *Verification features* (Gold SDK), or *Feature Set* (One Touch SDK). Verification features are what is compared to the Registration features when a user swipes a finger.

In these previous SDKs, you could not compare two Feature Sets, you could only compare a Feature Set against a Fingerprint Template. The DigitalPersona Biometric SDK removes the distinction between feature sets and templates when using the ANSI/ISO formats. For data created in ANSI/ISO data formats, all minutiae data is stored in an FMD, whether produced by the feature extraction functions or the enrollment functions.

## 5.5 Changes in Terminology from Previous Usage (Gold and One Touch)

This section describes differences in terminology between this DigitalPersona Biometric SDK and the One Touch and Gold documentation.

Old Term	New Term	Explanation
Fingerprint authentication	Fingerprint verification	Change in industry standard terminology.
Fingerprint registration	Fingerprint enrollment	Change in industry standard terminology.
Fingerprint sensor (referring to a fingerprint capture device)	Fingerprint reader	Erroneous usage. Sensors are a <i>component</i> of certain types of fingerprint capture devices.

Old Term	New Term	Explanation
Match Matching Matching score	Compare Comparison Comparison score	Change in industry standard terminology.
Performance	Recognition accuracy	More accurate terminology.
Fingerprint feature set Fingerprint template	Fingerprint minutiae data	Fingerprint templates (fingerprint features stored for enrolled fingers) and fingerprint feature sets (images used for verification and identification) have been replaced with fingerprint minutiae data in the U.are.U/Digital Persona family of SDKs only. With standards-based data formats in the U.are.U/DigitalPersona Biometric SDKs, all fingerprints are stored in the same format.
ROC curve	DET curve	More accurate terminology. We have never used ROC curves, however our DET curves were sometimes erroneously called ROC curves in the past.
FAR	FMR or FNMR as appropriate	The definition of FAR is application-specific. In some applications FAR is similar to FMR while in other applications, FAR is similar to FNMR.
FRR	FMR or FNMR as appropriate	The definition of FRR is application-specific. In some applications FRR is similar to FMR while in other applications, FRR is similar to FNMR.

## 5.6 Upgrading from Digital Persona UPOS for OPOS/JavaPOS

This new DigitalPersona Biometric SDK version of the OPOS and JavaPOS APIs are backward-compatible. However the following new features may or may not affect your existing applications:

- *The default data format has a longer header.* The default data format is now fully compliant with UnifiedPOS 1.13. The default data format is the same as the old format except that the template header has additional bytes of information, i.e., a 45-byte header, instead of the previous 10-byte header for JavaPOS and 12-byte header for OPOS. If you do not update your application to specify a specific format, new templates will be created with 45-byte headers. Enrollment, identification and verification will continue to work with both new templates with 45-byte headers and previous templates that have the shorter headers. See [Section 13.2 Working with Fingerprint Data in JavaPOS](#) and [Section 14.2 Working with Fingerprint Data in OPOS](#) for more information.
- *New FAR Security Setting.* We have tightened the FAR security setting to align with our current corporate standard. The new standard is 100 times more strict. Please update your code accordingly. We recommend that you allow the FAR setting to be configurable at run-time within your application. See [Section 3.5.4 Determining an Acceptable Level of Error](#) for more information on setting FAR.
- *Enrollment no longer stops at four retries.* The previous enrollment process would fail after four unsuccessful scans. If your application is dependent on having exactly four tries, then you may need to adjust your code.

- *Errors and exceptions may be handled differently.* Because of the extensive architectural changes to this version of the OPOS and JavaPOS APIs, exceptions and error codes may not be identical. You should double check carefully that the new API handles errors and exceptions as expected for your application.

For specific details about upgrading your existing application, consult [Section 13 The JavaPOS API](#) or [Section 14 The OPOS API](#).



# Chapter 6

## 6 Using the SDK

---

### 6.1 What's In the SDK?

The SDK consists of:

1. C/C++ API -- C libraries that conform to ANSI.C99 (<http://en.wikipedia.org/wiki/C99>):
  - DP Capture API - for capturing fingerprints
  - FingerJet Engine API - for extracting fingerprint characteristics and identifying/verifying fingerprints. Note that the FingerJet Engine API is not available on iOS,
2. .NET API -- .NET class libraries
  - DP .NET API - for capturing and comparing fingerprints
  - DP .NET Controls - simple interface for enrollment and identification, based on OneTouch interface
  - DP ActiveX Library - for capturing and comparing fingerprints using the ActiveX wrapper with .NET
  - DP ActiveX Controls - simple interface for enrollment and identification, based on OneTouch interface
3. ActiveX API and controls -- ActiveX class libraries
  - DPXUru.dll - ActiveX API library
  - DPctlXUru.dll - ActiveX GUI controls
4. Java API -- Java class libraries
  - dpuaru.jar - library classes and interfaces for working with readers and the FingerJet Engine
5. JavaScript API -- JavaScript files implementing the Fingerprint.WebApi object for working with fingerprint readers.
6. JavaPOS API -- class libraries that implement a JavaPOS-compliant API (per the JavaPOS 1.13 specification), as a wrapper to the Digital Persona Java API:
  - dpjavapos.jar - library classes and interfaces for working with readers
  - JavaPOS Device Service object, which can be used with any JavaPOS Device Control for the Biometrics device category
7. OPOS API -- classes that implement an OPOS-compliant API (per the UPOS 1.13 specification), as a wrapper to the Digital Persona C/C++ API:
  - dpServiceObject.dll - a custom implementation of the OPOS data service
  - OPOSBiometrics - a custom implementation of the OPOS biometrics control
8. Objective-C API -- classes for working with fingerprint readers. Note that the FingerJet Engine is not available on iOS,

9. Run-time components:

- Capture driver and SDK layer
- FingerJet Engine run-time

10. Sample applications in each language that demonstrate SDK features.

## 6.2 FingerJet Engine

The FingerJet Engine is a module that extracts fingerprint characteristics from image records to create FMDs and compares FMDs to confirm identity. The FingerJet Engine has met the PIV performance thresholds for fingerprint minutiae data generation required by NIST.

We support a maximum of 16 views in a single FID or FMD during authentication. All views in a single record must have the same resolution. We do not support unknown finger positions (finger position = 0).

Note that the FingerJet Engine is not available on iOS,

# Chapter 7

## 7 The C++ APIs

This chapter provides an overview of the C/C++ APIs available for the Windows and Linux platforms. For details on using the API with a specific OS platform, consult the corresponding Platform Guide.

Detailed documentation for the C++ API is contained in the header files. You can simply read the header files or you can view the Doxygen files. Consult the platform guide for details of where the Doxygen files are located.

The API is thread-safe.

### 7.1 DP Capture API

The DP Capture API consists of library management, reader management, capturing and streaming.

#### 7.1.1 Library Management

Function	Description
dpfpdd_init	Initialize the library (allocate system resources and initialize data). This must be the first function called.
dpfpdd_exit	Release the library and its resources.
dpfpdd_version	Query the library version. This is the only function that can be called before dpfpdd_init or after dpfpdd_exit.  This returns the DP Capture API library version (not the U.are.U SDK version). This is analogous to the dpfj_version function which returns the version of the FingerJet library file.

#### 7.1.2 Fingerprint Capture Device Management

Function	Description
dpfpdd_query_devices	Discover connected devices.
dpfpdd_open	Open a device. This function establishes an exclusive link to the device; no other processes will be able to use the device until you close it. The application <i>must</i> open the device before use.
dpfpdd_close	Close a device.
dpfpdd_get_device_status	Get the status for a device. You would normally check the device status between captures to ensure that the device is functioning and there are no error conditions.
dpfpdd_get_device_capabilities	Query a device for information on capabilities.

Function	Description
dpfpdd_set_parameter	Change a device or driver parameter. This function is not supported in the TouchChip Device Add-On for Android.
dpfpdd_get_parameter	Query a device or driver parameter. This function is not supported in the TouchChip Device Add-On for Android.
dpfpdd_led_config	Sets operation mode for LED: automatic or controlled by client application. This function is not supported in the TouchChip Device Add-On for Android.
dpfpdd_led_ctrl	Changes reader or driver setting. This function is not supported in the TouchChip Device Add-On for Android.
dpfpdd_calibrate	Calibrate a reader. Some readers are self-calibrating. Ambient light or temperature can affect calibration, for some readers. Calibration can take several seconds. This function is not supported in the TouchChip Device Add-On for Android.
dpfpdd_reset	Do a hardware reset on the reader. Hardware resets are typically needed only after a hardware problem (e.g., the device is unplugged or receives an electrostatic shock). Hardware resets typically only take a few milliseconds. This function is not supported in the TouchChip Device Add-On for Android.

## 7.1.3 Capturing Fingerprints

The dpfpdd\_capture function captures a fingerprint image for

- Enrollment (as part of the process described on page 27)
- Identifying users with dpfj\_identify
- Verifying a specific user identity with dpfj\_compare

Function	Description
dpfpdd_capture	Capture a fingerprint image from the reader. This function signals the reader that a fingerprint is expected, and waits until a fingerprint is received.
dpfpdd_capture_async	Starts asynchronous capture on the opened reader device, signaling that a fingerprint is expected and then exits. This function is not supported in the TouchChip Device Add-On for Android.
dpfpdd_capture	Capture a fingerprint image from the reader. This function signals the reader that a fingerprint is expected, and waits until a fingerprint is received.

**Note:** In order to comply with Apple External Accessory power requirements, capture within iOS is limited to 20 seconds per capture call.

## 7.2 FingerJet Engine API

The FingerJet API contains functions that extract features from FIDs to create FMDs, identify/verify FMDs and convert FMDs to different formats.

Note: The FingerJet Engine is not available on iOS.



## 7.2.1 Library Management

Function	Description
dpfj_version	Query the library version. This returns the FingerJet library version (not the U.are.U SDK version). This is analogous to the dpfpdd_version function which returns the version of the DP Capture API library file.

## 7.2.2 Select Fingerprint Matching Engine

Function	Description
dpfj_select_engine	<p>FingerJet<sup>(R)</sup> is the default engine used if this function is not called. The FingerJet engine is available on all platforms and does not require open reader (parameter hdev can be NULL). Not every other engine is available on every platform. Some engines require a valid handle from opened reader to be supplied.</p> <p>Currently, you can choose from two types of FingerJet engine, versions 6 and 7. On some platforms the Innovatrics engine is also available. If this function is not called and no choice is made by the application, the default matching engine used is FingerJet version 6. See also <a href="#">Section 16.2 Minex certification</a>.</p>

## 7.2.3 Extract FMD

Function	Description
dpfj_create_fmd_from_raw	Creates FMD from raw image
dpfj_create_fmd_from_fid	Creates FMD from ANSI, ISO or Digital Persona legacy format FID

## 7.2.4 Identify Fingerprint

This function is detailed in [Section 3.4 Workflow - Identifying/Verifying](#)

Function	Description
dpfj_identify	Identify an FMD: given an array of FMDs, this function returns an array of candidates that match the original fingerprint (an FMV within an FMD) within the threshold of error. Supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.

## 7.2.5 Enrollment

The enrollment functions allow you to enroll a finger to create an FMD that you can store in your database. For ANSI/ISO formats, the enrollment functions create FMDs. For the legacy Digital Persona format, the enrollment functions create a fingerprint template.

The typical process would be:

1. Call dpfj\_start\_enrollment.
2. Capture a fingerprint scan and extract an FMD, using the standard functions (dpfpdd\_capture to capture and dpfj\_create\_fmd\_from\_fid or dpfj\_create\_fmd\_from\_raw to extract).
3. Call dpfj\_add\_to\_enrollment to add the fingerprint to the potential pool.
4. Repeat the previous two steps until dpfj\_add\_to\_enrollment returns a flag indicating the pool of FMDs is now sufficient to create an enrollment FMD.

5. Create the enrollment FMD with `dpfj_create_enrollment_fmd` and release resources by calling `dpfj_finish_enrollment`.
6. Store the enrollment FMD in your database. Some applications like voting, banking and law enforcement require that you check for duplicate fingerprints before storing a new fingerprint in the database.

Function	Description
<code>dpfj_start_enrollment</code>	Begin the enrollment process and allocate resources.
<code>dpfj_add_to_enrollment</code>	Add the FMD to the pool of FMDs for enrollment and return a flag indicating that the enrollment is ready (enough FMDs have been received to create the enrollment FMD)
<code>dpfj_create_enrollment_fmd</code>	Create FMD for enrolled finger
<code>dpfj_finish_enrollment</code>	Release resources used during enrollment process

## 7.2.6 Format conversion

Function	Description
<code>dpfj_fmd_convert</code>	Convert FMDs from ANSI to ISO format and vice versa.
<code>dpfj_dp_fid_convert</code>	Convert legacy Digital Persona image (Gold SDK and One Touch SDK) to ANSI or ISO images

## 7.2.7 Advanced Diagnostics

The majority of applications should use the `dpfj_identify` function to implement both identification and verification. However, in a few special cases, e.g., using multi-modal biometrics, or doing statistical risk assessment, the `dpfj_compare` function allows you to compare two FMVs to determine their actual degree of dissimilarity. This is useful for accuracy testing and diagnostics and is not intended to be used in final applications for actual fingerprint recognition. The `dpfj_compare` function returns a dissimilarity score with values:

- 0 = fingerprints are NOT dissimilar (i.e., they MATCH perfectly).
- `maxint` (`#7FFFFFFF` or `2147483647`) = fingerprints are completely dissimilar (i.e., DO NOT match).
- Values close to 0 indicate very close matches, values closer to `maxint` indicate very poor matches.

The table below shows the relationship between the scores returned from `dpfj_compare` and the false match error rates observed in our test. The dissimilarity score distribution is estimated based on our internal testing, and may not be representative of the actual rate that will be observed in deployment.

Dissimilarity Score	False Match Rate
2147483	.1%
214748	.01%
21474	.001%
2147	.0001%

Function	Description
<code>dpfj_compare</code>	Compare two FMDs; supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.

## 7.2.8 Wavelet Scalar Quantization (WSQ) Compression

The DigitalPersona Biometric SDK provides compression of fingerprint images with the WSQ algorithm. This compression is available in two ways:

1. Using code developed by NIST. This code is included within the Digital Persona C/C++ API.
2. Using the WSQ1000 SDK from Aware, Inc. on the target system. If the WSQ 1000 SDK is installed, the Digital Persona functions will call the WSQ 1000 functions to do compression and decompression instead of the native NIST-based code. However, see additional information on “Raw” WSQ compression on page 30.

Note that HID Global does not redistribute the Aware WSQ1000 SDK, it must be acquired and installed separately.

The typical sequence to compress an image would be:

```
dpfj_start_compression
dpfj_set_wsq_bitrate or dpfj_set_wsq_size - Set the desired size for the compressed image
dpfj_compress_fid or dpfj_compress_raw - Compress the FID or raw image
dpfj_get_processed_data - Retrieve the compressed image
dpfj_finish_compression
```

The typical process to decompress an image would be:

```
dpfj_start_compression
dpfj_expand_fid or dpfj_expand_raw - Decompress an FID or raw image
dpfj_get_processed_data - Retrieve the expanded image
dpfj_finish_compression
```

Function	Description
dpfj_finish_compression	Releases resources.
dpfj_set_wsq_bitrate dpfj_set_wsq_size	These two specify the same thing: the size of the resulting compressed image, and can be used interchangeably.  Setting the bitrate at 0.75 bpp to 0.9 bpp allows compression of 15:1 to 12:1.  The parameter <code>tolerance_aw</code> sets the tolerance as required by the Aware WSQ1000 SDK and it is ignored when using NIST algorithm.
dpfj_compress_fid dpfj_compress_raw	Compress an FID or raw image, according to the requested size.
dpfj_get_processed_data	Retrieve the image that was just compressed/expanded.
dpfj_expand_fid dpfj_expand_raw	Expand a previously compressed FID or raw image.

## 7.2.9 Raw WSQ Compression

In the majority of situations where WSQ data is a solution requirement, the WSQ data is expected to be in the “raw” format. What this means is that the data contains pure WSQ specification data and no ISO/ANSI header. Below is a code snippet showing how to generate a raw WSQ file from a captured fingerprint image data object. The generated file can subsequently be opened in most common WSQ viewer applications.

```
DPUruNet.Compression.Start();
DPUruNet.Compression.SetWsqBitrate(90, 0);

/* Using Raw compression */
```

```
Fid ISOFid = captureResult.Data; //captureResult is a parameter passed into the Async capture
listener\callback and contains the captured fingerprint image data (Fid).
byte[] rawCompress = DPURunet.Compression.CompressRaw(ISOFid.Views[0].Width,
ISOFid.Views[0].Height, 500, 8, ISOFid.Views[0].RawImage,
CompressionAlgorithm.COMPRESSION_WSQ_NIST);
/* Creates valid WSQ file */
File.WriteAllBytes("WSQfromRaw.wsq", rawCompress);
```

## 7.2.10 NIST Fingerprint Image Quality (NFIQ)

The DigitalPersona Biometric SDK calculates NFIQ scores for fingerprint images. This calculation is available in two ways:

1. Using code developed by NIST. This code is included within the Digital Persona C/C++ API.
2. Using the WSQ1000 SDK from Aware, Inc. on the target system. If the WSQ1000 SDK is installed, the Digital Persona functions will call the WSQ1000 functions to do NFIQ calculation instead of the native NIST-based code.

Note that HID Global does not redistribute the Aware WSQ1000 SDK, it must be acquired and installed separately.

Function	Description
dpfj_quality_nfiq_from_fid	Calculate the NFIQ score of an FID or raw image.
dpfj_quality_nfiq_from_raw	NFIQ scores range from 1 to 5, with 1 being the best quality and 5 indicating that the image is not suitable for feature extraction.



# Chapter 8

## 8 The Java API

---

The Java API is built as a wrapper to the C/C++ API. The Java API is available for Linux, Android and Windows. This chapter provides an overview of the API. For details of using the API on a specific reader platform, consult the appropriate Platform Guide.

The Java API is considerably simpler to use than the C/C++ APIs and therefore generally results in:

- Easier data management
- Easier enrollment
- Faster development

### 8.1 Importing the Digital Persona Java package

The Digital Persona Java library classes and interfaces are aggregated into `dpuaru.jar`. To use the Digital Persona Java library functionality import the `com.Crossmatch.uareu.*` package, and make sure to include `dpuareu.jar` into your classpath.

### 8.2 Getting Detailed Documentation

This chapter provides an overview of the main methods in the Java API. For a complete description of method parameters, the Javadoc documentation for the Java libraries is provided. Consult the platform guide for details of where the Javadoc files are located.

### 8.3 Using the Package

#### 8.3.1 Main Access Point

The main access point to the Digital Persona Java library is the `UareUGlobal` class. This is a static class, which allows you to acquire references to the classes related to fingerprint readers and to the FingerJet Engine:

- To acquire a reference to `ReaderCollection` use the `GetReaderCollection()` method. To destroy `ReaderCollection`, (release all system resources associated with readers and make readers available for other processes) use the `DestroyReaderCollection()` method.
- To acquire a reference(s) to individual readers, use the `ReaderCollection` object, which is a collection of objects of type `Reader`.
- To acquire a reference to the FingerJet Engine use the `GetEngine()` method. The engine does not use or allocate any system resources except memory and does not have to be destroyed explicitly.

#### 8.3.2 UareUException

The `UareUException` interface describes exceptions specific to the DigitalPersona Biometric SDK.

### 8.3.3 Getting a List of Available Readers

The ReaderCollection interface provides a list of the readers connected to the machine. A list of available readers can be acquired any time with GetReaders() method.

## 8.4 Working with Readers

Each attached reader is represented with a Reader object. The Reader interface allows:

- Querying reader description and capabilities,
- Acquiring status of the reader,
- Capturing fingerprints,
- Starting and stopping video stream, and
- Resetting and calibrating the reader.

The main methods are:

Function	Description
GetDescription	Get the description of a reader. The description is available at any time (even if the device is not open). This is the only method that can be called before the open () method.  Returns an object of type Description holding information about the reader hardware.
Open	Open a device and return the device capabilities. This method establishes an exclusive link to the device; no other processes will be able to use the device until you close it. The application <i>must</i> open the device before use.
GetStatus	Get the status for a device. You would normally check the device status between captures to ensure that the device is functioning and there are no error conditions.  Returns an object of type ReaderStatus which describes the current status of the reader.
GetCapabilities	Get the capabilities of a device.  Returns an object of type Capabilities which describes what the reader can do.
Calibrate	Calibrate a device. Some devices are self-calibrating. Ambient light or temperature can affect calibration, for some devices. Calibration can take several seconds.
Reset	Do a hardware reset on the reader. Hardware resets are typically needed only after a hardware problem (e.g., the device is unplugged or receives an electrostatic shock). Hardware resets typically only take a few milliseconds.
Close	Close a reader and release the resources associated with the reader.

### 8.4.1 Capturing Fingerprints

The Capture function captures a fingerprint image for

- Enrollment (as part of the process described in [Section 3.3 Workflow - Enrollment Application](#)).
- Identifying users with Identify

- Verifying a specific user identity with Compare

The primary fingerprint capture methods are:

Function	Description
Capture	Captures a fingerprint image from the open reader. This function signals the reader that a fingerprint is expected, and blocks until a fingerprint is received, capture fails or the reader times out.
CancelCapture	Cancel a pending capture

### 8.4.2 Streaming Fingerprints

Not all readers support streaming mode. To determine if a specific reader supports this feature, get the reader capabilities with the GetCapabilities method and check the value of the field can\_stream.

The streaming methods are:

Function	Description
StartStreaming	Put the reader into streaming mode. In this mode, the application must call GetStreamImage() to acquire images from the stream.
GetStreamImage	Capture a fingerprint image from the streaming data.  After this function returns, the reader remains in streaming mode.  Frame selection, scoring and other image processing are not performed by this function.
StopStreaming	End streaming mode

## 8.5 Accessing the FingerJet Engine

The Engine interface provides functionality to

- Extract fingerprint features (create FMDs),
- Identify and compare FMDs, and
- Create enrollment FMDs.

See [Section 3.2 Understanding the Data Flow](#) for details on enrollment and comparison terminology and data flow.

### 8.5.1 Creating FMDs from images

The CreateFmd() method can be used in two ways:

1. Extract fingerprint minutiae from a raw image and create an FMD.
2. Extract fingerprint minutiae from an FID and create an FMD.

The following limitations are applied to the raw images and FIDs:

- 8 bits per pixel
- no padding
- square pixels (horizontal and vertical dpi are the same)

The size of the resulting FMD will vary depending on the minutiae in a specific fingerprint.

## 8.5.2 Identification and Comparison

### 8.5.2.1 Identify()

Identifies a single FMD against an array of FMDs. This function takes as inputs:

- A single view in an FMD
- An array of FMDs (each FMD can contain up to 16 views) to compare
- The desired number of candidates to return
- The threshold for False Positive Identification Rate (FPIR) that is permitted

Each time a view has a score lower than the threshold FPIR, that view is marked as a possible candidate. Then when all possible candidates are identified (i.e., they meet the threshold), they are ranked by their score. Finally, the function returns as many candidates as requested, based on the candidates with the lowest dissimilarity score. For a discussion of setting the threshold as well as the statistical validity of the dissimilarity score and error rates, consult [Section 3.5.3 NIST Fingerprint Image Quality \(NFIQ\)](#).

### 8.5.2.2 Compare()

Takes two single views from two FMDs and returns a dissimilarity score indicating the quality of the match.

The majority of applications should use the Identify method to implement both identification and verification. However, in a few special cases, e.g., using multi-modal biometrics, or doing statistical risk assessment, the Compare method allows you to compare two FMVs to determine their actual degree of dissimilarity. This is useful for accuracy testing and diagnostics and is not intended to be used in final applications for actual fingerprint recognition. The Compare method returns a dissimilarity score with values:

- 0 = fingerprints are NOT dissimilar (i.e., they MATCH perfectly).
- maxint (#7FFFFFFF or 2147483647) = fingerprints are completely dissimilar (i.e., DO NOT match).
- Values close to 0 indicate very close matches, values closer to maxint indicate very poor matches.

The table below shows the relationship between the scores returned from Compare and the false match error rates observed in our test. The dissimilarity score distribution is estimated based on our internal testing, and may not be representative of the actual rate that will be observed in deployment.

Dissimilarity Score	False Match Rate
2147483	.1%
214748	.01%
21474	.001%
2147	.0001%

Function	Description
Compare	Compare two FMDs; supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.
Identify	Identify an FMD: given an array of FMDs, this function returns an array of candidates that match the original fingerprint (an FMV within an FMD) within the threshold of error. Supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.



### 8.5.3 Enrollment

CreateEnrollmentFmd() creates and returns an enrollment FMD. It takes as input a reference to an object of type EnrollmentCallback. The client must implement EnrollmentCallback.GetFmd(). This method acquires and returns an FMD to add to the enrollment. The engine calls EnrollmentCallback.GetFmd() as many times as needed in order to create an enrollment FMD.

Normally, the client application will implement EnrollmentCallback.GetFmd() to provide the onscreen UI to capture fingerprints from the reader, extract features using CreateFmd(), and return an enrollment FMD. If the user wants to cancel the enrollment, EnrollmentCallback.GetFmd() should return null.





# Chapter 9

## 9 The Javascript API

---

The JavaScript API, and an included JavaScript-based sample application, are provided as part of the DigitalPersona Biometric SDK for Windows 3.0 and above.

The API provides web-based capture of fingerprints on the Windows platform through the following browsers: Internet Explorer, Microsoft Edge, Google Chrome or Mozilla Firefox. Note that it provides capture only, and does not provide authentication or identification.

This chapter provides a complete description of the methods, events and enumerations of the JavaScript API. For details on installation, setting up the JavaScript development environment, and using the sample application, consult the DigitalPersona Biometric SDK Windows Platform Guide (version 3.0 or later).

### 9.1 Overview

The JavaScript API allows web applications to conveniently and securely acquire fingerprint data from a supported fingerprint reader connected to the user's device. The web application can then use the captured fingerprint data for purposes of user biometric enrollment and verification. The API allows software developers to:

- Enumerate fingerprint readers
- Select a fingerprint reader to be used with fingerprint capture
- Get the characteristics of a fingerprint reader
- Start fingerprint capture using a selected fingerprint reader
- Stop fingerprint capture
- Receive captured fingerprints in the following formats: PNG image, WSQ, Intermediate and Raw.
- Receive activity notifications from the fingerprint reader
- Receive an indication of the quality of the fingerprint capture
- Monitor device connection and disconnection

### 9.2 Using the fingerprint library

The Fingerprint Library consists of two JavaScript files, referenced through use of the HTML `<script>` tag. In the example below, the ECMAScript 6 shim is also included.

```
<script src="es6-shim.js"></script>
<script src=" websdk.client.bundle.min.js"></script>
<script src=" fingerprint.sdk.min.js"></script>
```

All the functionality of the Fingerprint Library is accessed with the `Fingerprint.WebApi` object. You create an instance of `Fingerprint.WebApi` as shown below.

```
this.sdk = new Fingerprint.WebApi;
```

## 9.3 Fingerprint.WebApi methods

The table below lists all of the Fingerprint.WebApi methods.

Function	Description
enumerateDevices()	Returns the list of fingerprint reader identifiers.
getDeviceInfo()	Returns information about a fingerprint reader.
startAcquisition()	Starts fingerprint capture.
stopAcquisition	Stops fingerprint capture.

Each method returns an ECMAScript 6 Promise object. The fulfillment handler receives the result of the method, if any, and the rejection handler receives an instance of the standard Error object.

### 9.3.1 enumerateDevices()

The enumerateDevices() method returns the list of identifiers for all fingerprint readers connected to the computer.

#### TypeScript declaration

```
enumerateDevices(): Promise<string[]>;
```

#### Parameters - None

#### Returns

Returns an ES6 promise, which gets fulfilled if the list of the readers is received successfully, or rejected otherwise. The fulfillment handler receives an array of strings, each string contains the unique identifier of each reader.

Code Snippet - The following code will print out device UUIDs for all connected fingerprint readers.

```
var FingerprintSdkTest = (function () {
    function FingerprintSdkTest() {
        this.sdk = new Fingerprint.WebApi;
    }
    FingerprintSdkTest.prototype.getDeviceList = function () {
        return this.sdk.enumerateDevices();
    };
    return FingerprintSdkTest;
})();

window.onload = function () {
    test = new FingerprintSdkTest();
    var allReaders = test.getDeviceList();
    allReaders.then(function (successObj) {
        for (i=0;i<successObj.length;i++){
            console.log(successObj[i]);
        }
    }, function (error){
        console.log(error.message);
    });
}
```

### 9.3.2 getDeviceInfo()

The getDeviceInfo() method returns information about a connected fingerprint reader.

## TypeScript declaration

```
getDeviceInfo(deviceUid: string): Promise<DeviceInfo>;
```

### Parameter

Parameter	Description
deviceUid	Required. The unique identifier of the fingerprint reader.

### Returns

Returns an ES6 promise, which gets fulfilled if the information about the reader is received successfully, or rejected otherwise. The fulfillment handler receives an instance of the DeviceInfo object, which will be described below.

## 9.3.3 DeviceInfo

The DeviceInfo object contains information about the fingerprint reader.

Parameter	Description
deviceId	A string containing the unique identifier of the fingerprint reader.
eUidType	One of the DeviceUidType enumeration values that specifies the type of the unique identifier of the fingerprint reader.
eDeviceModality	One of the DeviceModality enumeration values that specifies the capture process used by the fingerprint reader.
eDeviceTech	One of the DeviceTechnology enumeration values that specifies the fingerprint reader technology.

### Code Snippet

The following code will print out device information for all connected fingerprint readers (enumerated devices).

```
var FingerprintSdkTest = (function () {
    function FingerprintSdkTest() {
        this.sdk = new Fingerprint.WebApi;
    }
    FingerprintSdkTest.prototype.getDeviceList = function () {
        return this.sdk.enumerateDevices();
    };
    FingerprintSdkTest.prototype.getDeviceInfoWithID = function (uid) {
        var _instance = this;
        return this.sdk.getDeviceInfo(uid);
    };
    return FingerprintSdkTest;
})();

window.onload = function () {
    test = new FingerprintSdkTest();
    var allReaders = test.getDeviceList();
    allReaders.then(function (sucessObj) {
        for (i=0;i<sucessObj.length;i++){
            printDeviceInfo(sucessObj[i]);
        }
    }, function (error){
```

```

        console.log(error.message);
    });
}

function printDeviceInfo(uid) {
    var myDeviceVal = test.getDeviceInfoWithID(uid);
    myDeviceVal.then(function (sucessObj) {
        console.log(sucessObj.DeviceID); //A string containing the unique identifier of the
        fingerprint reader.
        console.log(Fingerprint.DeviceTechnology[sucessObj.eDeviceTech]); // One of the
        DeviceUidType enumeration values that specifies the type of the unique identifier of the
        fingerprint reader.
        console.log(Fingerprint.DeviceModality[sucessObj.eDeviceModality]); // One of the
        DeviceModality enumeration values that specifies the capture process used by the fingerprint
        reader.
        console.log(Fingerprint.DeviceUidType[sucessObj.eUidType]); // One of the
        DeviceTechnology enumeration values that specifies the fingerprint reader technology.

    }, function (error) {
        console.log(error.message);
    });
}

```

### 9.3.4 StartAcquisition()

The startAcquisition() method starts fingerprint capture on the client computer.

#### TypeScript declaration

```
startAcquisition(sampleFormat: SampleFormat, deviceUid?: string): Promise<void>;
```

Parameter	Description
sampleFormat	Required. One of the SampleFormat enumeration values that specifies the format of fingerprint data to be returned.
deviceUid	Optional. The unique identifier of the fingerprint reader. If not specified, the capture is performed on all available fingerprint readers.

#### Returns

Returns an ES6 promise, which gets fulfilled if the fingerprint capture operation is started successfully, or rejected otherwise.

**Code Snippet** - The following code will start capture mode.

```

var FingerprintSdkTest = (function () {
    function FingerprintSdkTest() {
        this.sdk = new Fingerprint.WebApi;
    }

    FingerprintSdkTest.prototype.startCapture = function () {
        this.sdk.startAcquisition(Fingerprint.SampleFormat.PngImage).then(function () {
            console.log("You can start capturing !!!");
        }, function (error) {
            console.log(error.message);
        });
    };
};

```

```

        return FingerprintSdkTest;
    }) ();

    window.onload = function () {
        test = new FingerprintSdkTest();
        test.startCapture();
    }

```

### 9.3.5 stopAcquisition

The `stopAcquisition()` method stops the previously started fingerprint capture on the client computer.

#### TypeScript declaration

```
stopAcquisition(deviceUid?: string): Promise<void>;
```

Parameter	Description
deviceUid	Optional. The unique identifier of the fingerprint reader. If not specified, the capture on all available fingerprint readers will be stopped.

Returns an ES6 promise, which gets fulfilled if the fingerprint capture operation is stopped successfully, or rejected otherwise.

**Code Snippet** - The following code will stop capture mode.

```

var FingerprintSdkTest = (function () {
    function FingerprintSdkTest() {
        this.sdk = new Fingerprint.WebApi;
    }

    FingerprintSdkTest.prototype.stopCapture = function () {
        this.sdk.stopAcquisition().then(function () {
            console.log("Capturing stopped !!!");
        }, function (error) {
            showMessage(error.message);
        });
    };

    return FingerprintSdkTest;
})();

window.onload = function () {
    test = new FingerprintSdkTest();
    test.stopCapture();
}

```

## 9.4 Fingerprint.WebApi events

The table below lists the `Fingerprint.WebApi` events.

Parameter	Description
DeviceConnected	A fingerprint reader is connected to the client computer.
DeviceDisconnected	A fingerprint reader is disconnected from the client computer.
SamplesAcquired	Fingerprint data has been captured.

Parameter	Description
QualityReported	Fingerprint image quality data has been reported.
ErrorOccurred	An error occurred during fingerprint capture.
CommunicationFailed	An error occurred during communication with the DigitalPersona Biometric SDK

### 9.4.1 DeviceConnected

The DeviceConnected event is fired when a fingerprint reader is connected to the client computer.

Property	Description
deviceId	A string containing the unique identifier of the fingerprint reader.

### 9.4.2 DeviceDisconnected

The DeviceDisconnected event is fired when a fingerprint reader is disconnected from the client computer.

Property	Description
deviceId	A string containing the unique identifier of the fingerprint reader.

### 9.4.3 SamplesAcquired

The SamplesAcquired event is fired when fingerprint data is captured.

Property	Description
deviceId	A string containing the unique identifier of the fingerprint reader.
SampleFormat	One of the SampleFormat enumeration values that specifies the format of the returned fingerprint data.
samples	A string containing a serialized JSON array of fingerprint samples. This string can be base64url encoded and passed directly to the Altus Web server components (aka Altus Confirm) authentication service as the fingerprint credential.

### 9.4.4 QualityReported

The QualityReported event is fired when fingerprint quality advice is reported.

Property	Description
deviceId	A string containing the unique identifier of the fingerprint reader.
quality	One of the QualityCode enumeration values that specifies advice regarding the quality of the scanned fingerprint image.

### 9.4.5 ErrorOccured

The ErrorOccured event is fired when an error occurred during fingerprint capture.

Property	Description
deviceId	A string containing the unique identifier of the fingerprint reader.
error	A number containing the HRESULT error code.



### 9.4.6 CommunicationFailed

The CommunicationFailed event is fired when an error occurred during communication with the DigitalPersona Biometric SDK.

**Properties** - None.

#### Remarks

The CommunicationFailed event usually indicates that the required DigitalPersona Biometric SDK components are not running on the client computer. Typically, the end-user of your application should be given instructions on installing the required software.

## 9.5 Enumerations

### 9.5.1 DeviceUidType

The DeviceUidType enumeration lists types of unique identifiers for fingerprint readers.

```
enum DeviceUidType {  
    Persistent = 0,  
    Volatile = 1,  
}
```

### 9.5.2 DeviceModality

The DeviceModality enumeration lists possible types of fingerprint readers relating to their capture process.

```
enum DeviceModality {  
    Unknown = 0,  
    Swipe = 1,  
    Area = 2,  
    AreaMultifinger = 3,  
}
```

### 9.5.3 DeviceTechnology

The DeviceTechnology enumeration lists types of fingerprint reader technologies.

```
enum DeviceTechnology {  
    Unknown = 0,  
    Optical = 1,  
    Capacitive = 2,  
    Thermal = 3,  
    Pressure = 4,  
}
```

### 9.5.4 SampleFormat

The SampleFormat enumeration lists formats of fingerprint data supported by the SDK. See page 43 for descriptions and JSON representation of each format.

```
enum SampleFormat {  
    Raw = 1,  
    Intermediate = 2,  
    Compressed = 3,  
    PngImage = 5,  
}
```

### 9.5.5 QualityCode

The QualityCode enumeration lists possible responses relating to the quality of the scanned fingerprint image.

```
enum QualityCode {
    Good = 0,
    NoImage = 1,
    TooLight = 2,
    TooDark = 3,
    TooNoisy = 4,
    LowContrast = 5,
    NotEnoughFeatures = 6,
    NotCentered = 7,
    NotAFinger = 8,
    TooHigh = 9,
    TooLow = 10,
    TooLeft = 11,
    TooRight = 12,
    TooStrange = 13,
    TooFast = 14,
    TooSkewed = 15,
    TooShort = 16,
    TooSlow = 17,
    ReverseMotion = 18,
    PressureTooHard = 19,
    PressureTooLight = 20,
    WetFinger = 21,
    FakeFinger = 22,
    TooSmall = 23,
    RotatedTooMuch = 24,
}
```

## 9.6 Sample Format details

Fingerprint data can be exported in one of the following formats.

- Raw - Data format is a raw (unprocessed) biometric sample, also referred to as a Fingerprint Image in most biometric documentation.
- Intermediate - Data format is a partially processed biometric sample, also referred to as a Feature Set in most biometric documentation.
- Compressed - Data format is a fully processed and compressed (WSQ) biometric sample, also referred to as a Fingerprint Template in most biometric documentation.
- PNGImage - Data format is a .PNG image file.

JSON representations of each of the supported export formats are shown below.

### 9.6.1 Raw

```
{
  deviceId:"30323030-3661-6533-3764-393600000000"
  sampleFormat:1
  samples:"[{
    "Data":{"{
      "Compression":0,
      "Data":"8PDw88PDw8PDw8PDw8PDw8PA",
```

```
// Base64url encoded image
"Format":{
  "iHeight":403,
  "iWidth":200,
  "iXdpi":508,
  "iYdpi":508,
  "uBPP":8,
  "uDataType":1,
  "uImageType":2,
  "uPadding":2,
  "uPlanes":1,
  "uPolarity":2,
  "uRGBcolorRepresentation":0,
  "uSignificantBpp":8
},
"Header":{
  "DeviceId":1407938095516745728,
  "DeviceType":49264417346420736,
  "iDataAcquisitionProgress":100,
  "uDataType":1
},
"Version":1
}",
"Header":{
"Encryption":0,
"Factor":8,
"Format":{
  "FormatID":0,
  "FormatOwner":51
},
"Purpose":0,
"Quality":-1,
"Type":1},
"Version":1
}]"
type:"SamplesAcquired"
}
```

## 9.6.2 Intermediate

```
{
deviceUid:"30323030-3661-6533-3764-393600000000"
sampleFormat:2
samples:"[{
  "Data":"eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9",
  //Base64url encoded Feature Set
  "Header":{
    "Encryption":0,
    "Factor":8,
    "Format":{
      "FormatID":0,
      "FormatOwner":51
    },
    "Purpose":0,
    //0 - DP_PURPOSE_ANY, meaning "Data" field is
    //equivalent to DP_PRE_REG/DP_VERIFICATION FMD and
    //can be used for both fingerprint enrollment or
```

```

        //fingerprint verification/identification purposes.
        "Quality":-1,
        "Type":2
    },
    "Version":1
  ]]"
  type:"SamplesAcquired"
}

```

### 9.6.3 Compressed (WSQ)

```

{
  deviceId:"30323030-3661-6533-3764-393600000000"
  sampleFormat:3
  samples:"[{
    "Data":{"{
      "Compression":2,
      "Data": "_6DqAB6Tk1TPFHxf1Oj0_6E",
      // Base64url encoded image
      "Format":{
        "iHeight":403,
        "iWidth":200,
        "iXdpi":508,
        "iYdpi":508,
        "uBPP":8,
        "uDataType":1,
        "uImageType":2,
        "uPadding":2,
        "uPlanes":1,
        "uPolarity":2,
        "uRGBcolorRepresentation":0,
        "uSignificantBpp":8},
        "Header":{
          "DeviceId":1407938095516745728,
          "DeviceType":49264417346420736,
          "iDataAcquisitionProgress":100,
          "uDataType":1
        },
        "Version":1
      }",
    "Header":{
      "Encryption":0,
      "Factor":8,
      "Format":{
        "FormatID":0,
        "FormatOwner":51
      },
      "Purpose":0,
      "Quality":-1,
      "Type":1
    },
    "Version":1
  ]]"
  type:"SamplesAcquired"
}

```

### 9.6.4 PNG

```
{  
  deviceId:"30323030-3661-6533-3764-393600000000"  
  sampleFormat:5  
  samples:[" sc7ZGpUWzaqVwDQ0e3qjwEjX7evPn_8PNES8Q6co2hUAAAAASUVORK5CYII"] "  
  // Base64url encoded image  
  type:"SamplesAcquired"  
}
```



# Chapter 10

## 10 Integration with web applications

The intended audience for this chapter are developers attempting to integrate biometric authentication within the web application (at login and authentication pages). Prior to reading this document it is strongly recommended to read the first six chapters of this guide and the chapter describing functions as implemented in your native API.

The web application will use two parts of the DigitalPersona Biometric SDK - the Javascript API and the native APIs (available in Java/.NET/C). During deployment the Digital Persona RTE must be distributed and installed at the client systems. The web application server will have the Digital Persona RTE also installed.



The Javascript API is demonstrated via the sample provided by the SDK and is called "UareUSampleWEB". This sample shows how to interact with a connected fingerprint reader for purposes of capturing fingerprints. After capturing prints, the web application must submit the data to the server for further processing (mainly identification or verification). It is expected the developer will reference the additional DigitalPersona Biometric SDK samples that demonstrate how to perform biometric verification and identification.

The Javascript API allows the selection of 4 different capture formats.

- RAW
- INTERMEDIATE
- PNG
- WSQ

The below sections describe when and how to use the various biometric capture formats exposed via the Javascript API.

### 10.1 RAW

The RAW data format is unformatted fingerprint image data. The typical data size of the image depends on the capture hardware but in general is in the ballpark of 100KB-200KB. The image attributes are encoded in the Samples data structure where things like height, width, dpi, and bits-per-pixel are specified. The RAW format is only used if application requirements mandate it; otherwise the recommendation is to use the more

efficient INTERMEDIATE format. There are two primary scenarios where you would want to use the RAW data format.

- ANSI or ISO minutiae standards are mandated
- Images are mandated to be kept for a specified period of time, or sent to a third party

### 10.1.1 ANSI or ISO minutiae standards are mandated

If you are mandated to use only ANSI or ISO minutiae standards, when you receive the image data on the server you can use the Crossmatch SDK to create ISO or ANSI fingerprint minutiae data using the ImportRaw methods of the SDK.

To get the ISO or ANSI minutiae data:

1. Be sure to start the image acquisition in the JavaScript using the RAW format.

```
startAcquisition(Fingerprint.SampleFormat.Raw)
```

2. Base64URL decode the image data into a byte array.

The image data is the corresponding value field of the Sample's "Data" field, as highlighted below.

```
{ deviceId:"30323030-3661-6533-3764-393600000000" sampleFormat:1 samples:[{
  "Data":{" Compression":0, "Data":"8PDw88PDw8PDw8PDw8PDw8PA", // Base64url encoded
  image "Format":{" iHeight":403, "iWidth":200, "iXdpi":508, "iYdpi":508, "uBPP":8,
```

Note - It is the responsibility of the application to parse out the critical data such as height, width, dpi, image data, etc...) either in the client JavaScript or at the server. These values should never be hard-coded.

3. Create the minutiae data using the appropriate SDK method shown below.

Java	<b>Engine.CreateFmd</b> (byte[] b64DecodedImage, int width, int height, int resolution, int finger_position, int cbeff_id, <b>Fmd.Format</b> format)
.NET	<b>CreateFmdFromRaw</b> (byte[] rawImageData, int fingerPosition, int CbeffId, int width, int height, int resolution, Constants.Formats.Fmd formatOut)

### 10.1.2 Images are kept or sent to a third party

If you are mandated to keep images for a certain period of time, or to submit images to a 3rd party system, the image format can be any format, however it is the responsibility of the application to convert the raw image data into the desired image format (such as BMP).

To get the image in a standards format:

1. Base64URL decode the image data into a byte array.
2. Import the image using the SDK method shown below.

Java	Importer.ImportRaw(byte[] data, int width, int height, int dpi, int finger_position, int cbeff_id, Fid.Format out_format, int out_dpi, boolean rotate180)
.NET	Importing RAW image data is not supported with .NET.



## 10.2 INTERMEDIATE

The intermediate format is not fingerprint image data. Instead it is fingerprint minutiae data encoded in what is known as the Digital Persona intermediate format. The typical size of this data is around 300 bytes. This is the recommended data format for optimal performance thanks to its reduced size and improved biometric recognition performance.

### 10.2.1 Biometric enrollment

In order to use the INTERMEDIATE data within a biometric system for user biometric enrollment follow the below workflow:

1. Select the intermediate format in JavaScript when capturing.

```
startAcquisition(Fingerprint.SampleFormat.Intermediate)
```

2. When 4 or more samples are captured, submit the samples to the server. The server should then base64URL decode the 4 samples and import them as pre-registration FMD format data using one of the below DigitalPersona Biometric SDK methods.

Java	Importer.ImportFmd(byte[] data, Fmd.Format format, Fmd.Format out_format))
.NET	Importer.ImportFmd (byte[] fmdIn, Constants.Formats.Fmd formatIn, Constants.Formats.Fmd formatOut )

fmdIn - This is the base64URL decoded sample data captured via the JavaScript.

formatIn - This should specify DP\_PRE\_REGISTRATION.

formatOut - This should specify DP\_PRE\_REGISTRATION.

3. Use the Enrollment interface to create an FMD of type DP\_REGISTRATION.

For Java, the interface and method to use is Engine.CreateEnrollmentFmd.

For C#, the interface and method to use is Enrollment.CreateEnrollmentFmd.

Note - Be sure to specify the format as DP\_REGISTRATION.

4. Securely encrypt and store the FMD for later usage (during user verification).

It is up to the application to decide the appropriate level of security and encryption for storing the fingerprint minutia data. The actual data to protect and store is accessed via the below member or method:

Java	Fmd/getData()
.NET	fmd.Bytes

#### 10.2.1.1 User verification

In order to use the INTERMEDIATE format for user verification workflow the application will capture and create an FMD on the server of type DP\_VERIFICATION.

1. Base64URL decode the biometric image data received at the server.
2. Import the fingerprint minutiae data for matching via the below methods.

**Note** - Be sure to specify DP\_VERIFICATION for the format.

Java	Importer.ImportFmd(byte[] data, Fmd.Format format, Fmd.Format out_format))
.NET	Importer.ImportFmd (byte[] fmdIn, Constants.Formats.Fmd formatIn, Constants.Formats.Fmd formatOut )

fmdIn - This is the base64URL decoded sample data captured via the JavaScript.

formatIn - This should specify DP\_VERIFICATION.

formatOut - This should specify DP\_VERIFICATION.

3. Compare the newly created FMD to an FMD from the database using the Compare method.

**Note** - Be sure to pass the DP\_VERIFICATION FMD as fmd1 and the DP\_REGISTRATION FMD as fmd2.

## 10.3 PNG

The PNG (portable network graphics) image format is a common image format that supports lossless compression and can be used as an alternative to RAW in most cases. Virtually all development platforms contain libraries or frameworks that support PNG image data. Use of the PNG format follows the RAW workflows, meaning the data can still easily be used for biometric matching purposes. The application simply must convert the PNG back to raw 8-bit grayscale data and then use it accordingly as described in the RAW section.

Conversion of PNG image data to raw 8-bit grayscale is outside the scope of this documentation.

## 10.4 WSQ

The WSQ (Wavelet Scalar Quantization) is an image format compressed according to the WSQ 3.1 specification. The image is compressed at a ratio of 10:1. This format, similar to PNG, is used to avoid sending the RAW image data over the network. Unlike PNG which is easily decompressed by platform libraries, the application must use the DigitalPersona Biometric SDK or compatible NBIS tool to decompress WSQ data back to raw image data:

1. Base64URL decode the received image data contained in the JSON received by the server.
2. Use the DigitalPersona Biometric SDK's compression methods to decompress back to raw.

**Note** - the compression algorithm must be specified as:

CompressionAlgorithm.COMPRESSION\_WSQ\_NIST

Java	Compression.Start() //To start a decompression operation Compression.ExpandRaw(byte[] data, Compression.CompressionAlgorithm compression_alg) Compression.Finish()
.NET	Compression.Start() //To start decompression operation Compression.ExpandRaw(byte[] data, DPURNet.CompressionAlgorithm compression_alg) Compression.Finish()



At this point the application has the raw image data which can be used according to the workflows specified in the RAW section. Applications should only use the WSQ format if mandated. Otherwise the INTERMEDIATE format is always recommended.



## 11 The .NET API

---

The .NET API is built as a wrapper to the C/C++ APIs for use on the Windows platform. This chapter provides an overview of the API. For details of using the API on a specific reader platform, consult the corresponding Platform Guide.

The .NET API is considerably simpler to use than the C/C++ APIs and therefore generally results in:

- Easier data management
- Easier enrollment
- Faster development

The .NET API also implements additional features:

- Serialization
- Pre-built GUI controls for enrollment and identification to quickly get you started

### 11.1 Importing the Digital Persona .NET package

The Digital Persona .NET library classes are aggregated into the following class libraries:

- DP .NET API - for capturing and comparing fingerprints
- DP .NET Controls - simple interface for enrollment and identification, based on OneTouch interface

### 11.2 Getting Detailed Documentation

This chapter provides an overview of the main methods in the .NET API. For a complete description of method parameters, the Doxygen documentation for the .NET libraries is provided. Consult the platform guide for details of where the Doxygen files are located.

### 11.3 Using the Package

#### 11.3.1 Main Access Points

- To acquire a reference to ReaderCollection use the GetReaders() method. To destroy ReaderCollection, (release all system resources associated with readers and make readers available for other processes) use the Dispose() method.
- To acquire a reference(s) to individual readers, use the ReaderCollection object, which is a collection of objects of type Reader.
- To work with the FingerJet Engine, instantiate one of the classes: Comparison, Importer, FeatureExtraction or Enrollment. For example, to start enrolling fingerprints using the FingerJet engine, call the static method, Enrollment.CreateEnrollmentFmd(). This method takes as input an enumeration

that specifies the format the enrollment template should be in, e.g., ANSI or ISO. This method also takes an `IEnumerable<Fmd>` object. See the topic below, *Enumerables in the .NET Wrapper*, for more information on `IEnumerable<Fmd>`.

### 11.3.2 SDKException

The `SDKException` class describes exceptions specific to the DigitalPersona Biometric SDK.

### 11.3.3 Serialization

The .NET API provides the ability to convert an FMD or FID into a format that can be read serially. Serialization allows you to easily transmit and store data as byte strings, streams or XML. This allows you to transmit the data or save it to standard file systems. The XML format can be used within HTML for building browser-based applications.

To return the data to its original format, deserialization is also provided.

Note that serialized FMDs and FIDs include the version number of the .NET wrapper was used for the serialization. If you attempt to deserialize with an older version of the wrapper, the results may not be correct. So when deserializing, the .NET wrapper will throw an `SDKException` if the serialization was done using a later version of the wrapper.

Serialization in the .NET Wrapper occurs using the default `System.Xml.Serialization.XmlSerializer`. The name of the root element of the generated XML is the same as the object type. Each public member of the object is represented as an XML element. Raw byte data is encoded as a Base-64 string by `XmlSerializer`.

Deserialization uses the same `XmlSerializer`. The XML that was serialized is fed into `XmlSerializer` where the elements with values become object members with values of the same name as the element.

### 11.3.4 IEnumerableables in the .NET Wrapper

The .NET wrapper uses the `IEnumerable<T>` interface (as specified here: <http://msdn.microsoft.com/en-us/library/9eekhta0.aspx>). An `IEnumerable<T>` is a data structure of type `T` that can be iterated through. An array is one example of `IEnumerable<T>`, because it can be iterated, or looped through.

An advantage of `IEnumerable` is that many different data structures can be used instead of requiring a very specific type of data structure, such as a 1-dimensional array of type `T`.

Perhaps the most important advantage of `Enumerables` is that whatever is responsible for sending an `IEnumerable` to a method may use special semantics such as 'yield return' to return only a single item at a time. This helps performance, and also helps programmers simplify their code by allowing .NET wrapper methods to pull only what is necessary for the operation.

For example, `Enrollment.CreateEnrollmentFmd(IEnumerable<Fmd>)` can iterate through a user's FMDs until enough data is captured and then create an enrollment FMD, as opposed to having more data than is necessary. Each iteration can cause a capture workflow to occur which causes 'yield return' to return a single item to `Enrollment`.

NOTE: The yield return feature of `IEnumerable` is available in C# but NOT in VB.NET.

For more information on `IEnumerable<T>`, consult the Microsoft documentation pages listed below.

Function	Description
<code>IEnumerable&lt;T&gt;</code>	<a href="http://msdn.microsoft.com/en-us/library/9eekhta0.aspx">http://msdn.microsoft.com/en-us/library/9eekhta0.aspx</a>
Yield	<a href="http://msdn.microsoft.com/en-us/library/9k7k7cf0(v=vs.80).aspx">http://msdn.microsoft.com/en-us/library/9k7k7cf0(v=vs.80).aspx</a>

## 11.4 Working with Readers

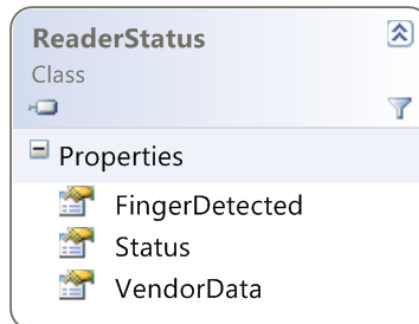
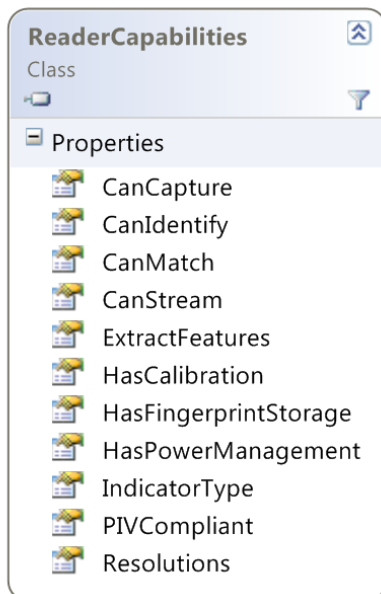
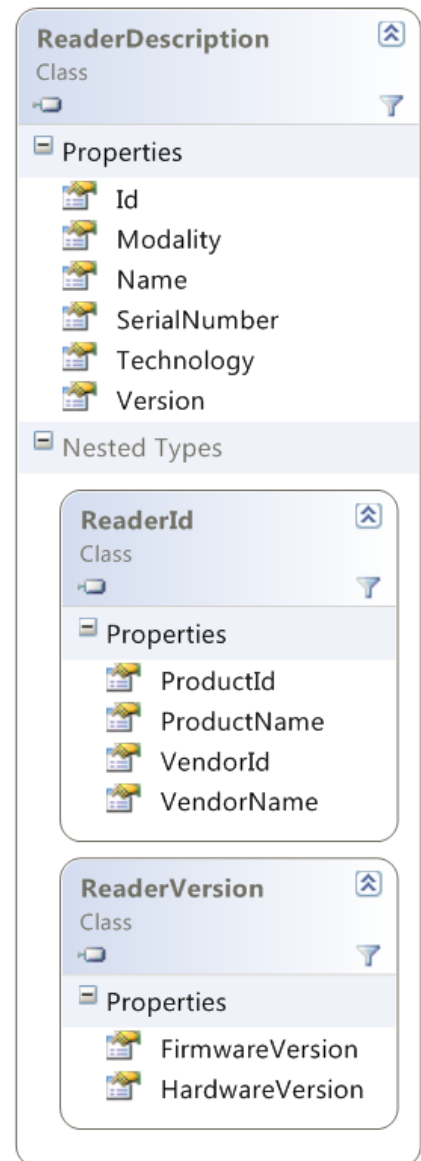
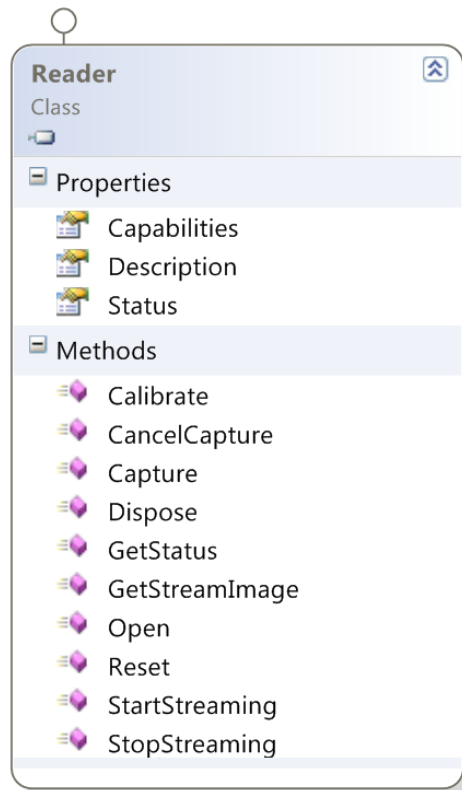
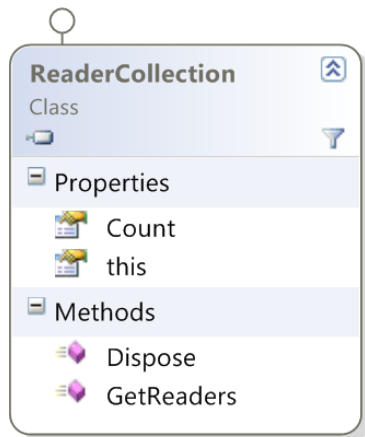
Readers are accessed through the ReaderCollection object, which is of type IEnumerable<Reader>.

Each attached reader is represented with a Reader object. The Reader interface allows:

- Querying reader description and capabilities,
- Acquiring status of the reader,
- Capturing fingerprints,
- Starting and stopping image stream, and
- Resetting and calibrating the reader.

To acquire a reader, call ReaderCollection.GetReaders() which returns the attached readers. Iterate through this list to look at the description object for the desired reader. Once finished with all of the Reader objects in the ReaderCollection, ensure that you call the Dispose() method to uninstantiate the ReaderCollection object and release system resources.

The UML diagrams below show the ReaderCollection and Reader classes.





The main methods for managing reader hardware in the Reader class are:

Function	Description
Open	Open a device and return the device capabilities. This method establishes an exclusive link to the device; no other processes will be able to use the device until you close it. The application <i>must</i> open the device before use.
GetStatus	Get the status for a device. You would normally check the device status between captures to ensure that the device is functioning and there are no error conditions.  Returns an object of type <code>ReaderStatus</code> which describes the current status of the reader.
Calibrate	Calibrate a device. Some devices are self-calibrating. Ambient light or temperature can affect calibration, for some devices. Calibration can take several seconds.
Reset	Do a hardware reset on the reader. Hardware resets are typically needed only after a hardware problem (e.g., the device is unplugged or receives an electrostatic shock). Hardware resets typically only take a few milliseconds.
Dispose	Close device, release memory, remove child objects.

### 11.4.1 Capturing Fingerprints

The Capture function captures a fingerprint image for

- Enrollment (as part of the process described in [Section 3.3 Workflow - Enrollment Application](#))
- Identifying users with Identify
- Verifying a specific user identity with Compare

The primary fingerprint capture methods are:

Function	Description
Capture	Captures a fingerprint image from the open reader. This function signals the reader that a fingerprint is expected, and blocks until a fingerprint is received, capture fails or the reader times out.
CancelCapture	Cancel a pending capture

### 11.4.2 Streaming Fingerprints

Not all readers support streaming mode. To determine if a specific reader supports this feature, check the value of the reader property `CanStream`.

The streaming methods are:

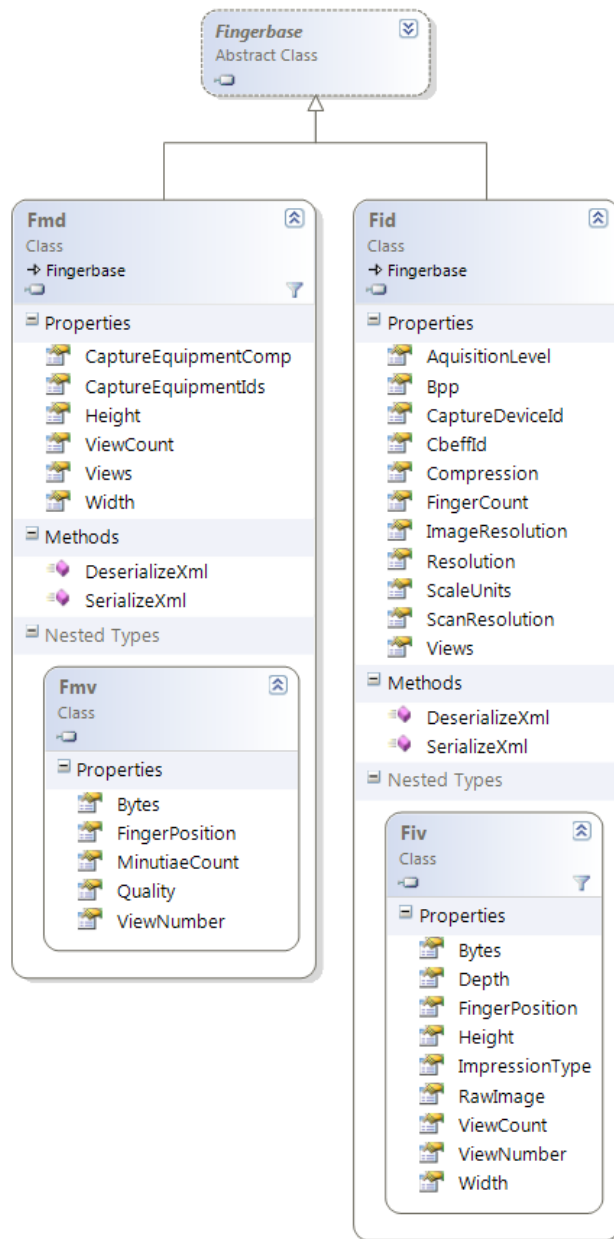
Function	Description
StartStreaming	Put the reader into streaming mode. In this mode, the application must call <code>GetStreamImage()</code> to acquire images from the stream.
GetStreamImage	Capture a fingerprint image from the streaming data.  After this function returns, the reader remains in streaming mode.  Frame selection, scoring and other image processing are not performed by this function.
StopStreaming	End streaming mode

## 11.5 Managing Fingerprint Data

The .NET API implements fingerprint data as shown in the UML diagram below and on the next page.

In addition to the usual methods for working with data, this class includes the methods below for serializing and deserializing (as described in [Section 11.1 Importing the Digital Persona .NET package](#)).

Function	Description
SerializeXml	Static method which serializes FMDs and FIDs into an XML string. Note that raw data is encoded as base-64.
DeserializeXml	Static method which deserializes XML-encoded FMDs and FIDs back into a Digital Persona FMD or FID data structure.



## 11.6 Analyzing and Managing Fingerprints (FingerJet Engine)

The FingerJet Engine interfaces provide functionality to

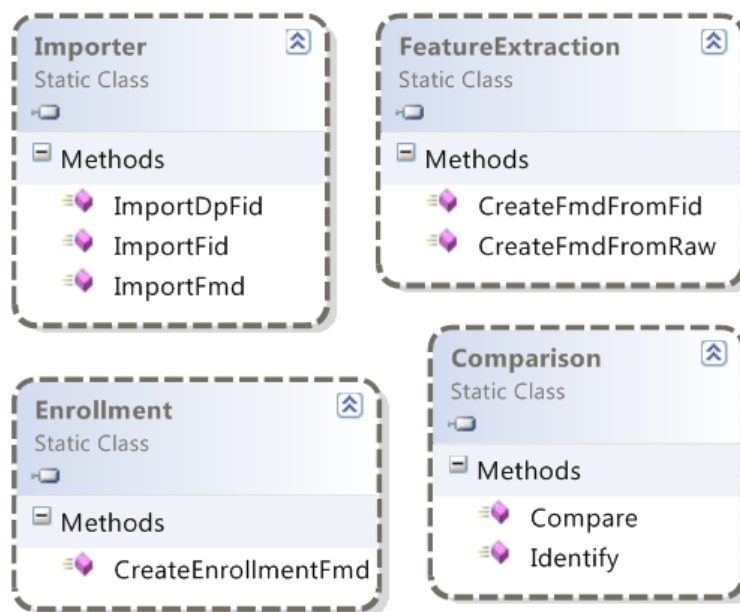
- Extract fingerprint features (create FMDs),
- Identify and compare FMDs, and
- Create enrollment FMDs.

Refer to [Section 3.2 Understanding the Data Flow](#) for a description of enrollment and comparison terminology and data flow.

The Classes for working with data (including identify and verify) are:

Function	Description
Comparison	This class allows you to perform: <ul style="list-style-type: none"> <li>Verification</li> <li>Identification</li> </ul>
Enrollment	To enroll a user, you must call: CreateEnrollmentFMD Enrollment is implemented as a static class -- you can provide an IEnumerable<Fmd> or a function that returns such.
Importer	The U.are.U SDK supports multiple formats for FMDs and FIDs. You can import any supported format.
FeatureExtraction	Extract features from an FID or raw image.

The UML diagram below provides more detail.



### 11.6.1 Creating FMDs from images

The CreateFmdFromFid() method can be used in two ways:

1. Extract fingerprint minutiae from a raw image and create an FMD.
2. Extract fingerprint minutiae from an FID and create an FMD.

The following limitations are applied to the raw images and FIDs:

- 8 bits per pixel
- no padding
- square pixels (horizontal and vertical dpi are the same)

The size of the resulting FMD will vary depending on the minutiae in a specific fingerprint.

### 11.6.2 Comparing Fingerprints

Identify() identifies a single FMD against an array of FMDs. This function takes as inputs:

- A single view in an FMD
- An array of FMDs (each FMD can contain up to 16 views) to compare
- The desired number of candidates to return
- The threshold for False Positive Identification Rate (FPIR) that is permitted

and returns matches as an array of integer pairs which provide the finger index and view index of each match.

Each time a view has a score lower than the threshold FPIR, that view is marked as a possible candidate. Then when all possible candidates are identified (i.e., they meet the threshold), they are ranked by their score. Finally, the function returns as many candidates as requested, based on the candidates with the lowest dissimilarity score. For a discussion of setting the threshold as well as the statistical validity of the dissimilarity score and error rates, consult [Section 3.5.3 NIST Fingerprint Image Quality \(NFIQ\)](#)

Compare() takes two single views from two FMDs and returns a dissimilarity score indicating the quality of the match.

The majority of applications should use the Identify method to implement both identification and verification. However, in a few special cases, e.g., using multi-modal biometrics, or doing statistical risk assessment, the Compare method allows you to compare two FMVs to determine their actual degree of dissimilarity. This is useful for accuracy testing and diagnostics and is not intended to be used in final applications for actual fingerprint recognition. The Compare method returns a dissimilarity score with values:

- 0 = fingerprints are NOT dissimilar (i.e., they MATCH perfectly).
- maxint (#7FFFFFFF or 2147483647) = fingerprints are completely dissimilar (i.e., DO NOT match).
- Values close to 0 indicate very close matches, values closer to maxint indicate very poor matches.

The table below shows the relationship between the scores returned from Compare and the false match error rates observed in our test. The dissimilarity score distribution is estimated based on our internal testing, and may not be representative of the actual rate that will be observed in deployment.

Dissimilarity Score	False Match Rate
2147483	.1%
214748	.01%
21474	.001%
2147	.0001%

Function	Description
Compare	Compare two FMDs; supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.
Identify	Identify an FMD: given an array of FMDs, this function returns an array of candidates that match the original fingerprint (an FMV within an FMD) within the threshold of error. Supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.

## 11.7 Enrollment

CreateEnrollmentFmd() creates and returns an enrollment FMD. The method takes as input an enumeration which defines which format to use, e.g., ANSI or ISO. This method also takes as input an IEnumerable<Fmd> object type. See [Section 11.3.4 IEnumerable<Fmd> in the .NET Wrapper](#) for a brief description. As IEnumerable<Fmd> pertains to Enrollment, you may either send an array of FMDs to CreateEnrollmentFmd() because simple arrays satisfy the IEnumerable<Fmd> interface. You may also send a method which returns IEnumerable<Fmd>. Here is an example that uses an IEnumerable<Fmd> method. To keep the following example simple, error checking has been removed:

```
// Create an enrollment Fmd.
DataResult<Fmd> enrollmentResult = Enrollment.CreateEnrollmentFmd(
    Constants.Formats.Fmd.ANSI,
    CaptureExtractFmd()
);
```

Here is a function which captures and extracts an FMD, then returns IEnumerable<Fmd>: (the full version of this code is in the code sample)

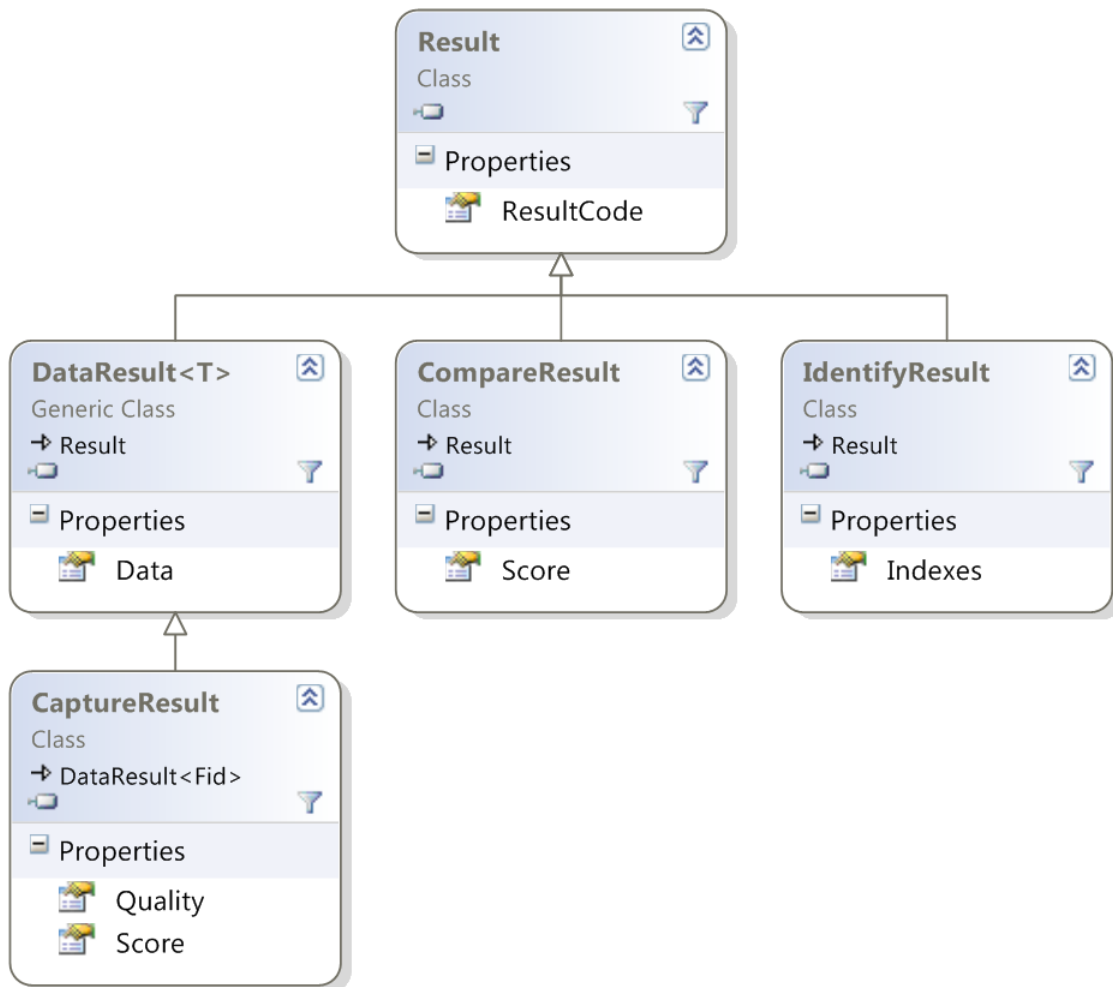
```
// Capture and extract an FMD and return as IEnumerable<Fmd>.
private IEnumerable<Fmd> CaptureAndExtractFmd()
{
    while (true)
    {
        // !!! Get Status and ensure that status is DP_STATUS_READY before continuing
        // Capture a fingerprint.
        CaptureResult captureResult = m_reader.Capture(
            Constants.Formats.Fid.ANSI,
            Constants.CaptureProcessing.DP_IMG_PROC_DEFAULT,
            /* Default processing used. */
            -1,
            /* No timeout. */
            reader.Capabilities.Resolutions[0]
            /* A resolution that is available to reader. */
        );

        // !!! Check for errors, use 'yield return null; or break;' to stop.
        yield return convertResult.Data;
    }
}
```

In VB.NET, the IEnumerable<T> class does not exist. The recommended method to enroll FMDs using VB.NET is to send CreateEnrollmentFmd() four captured FMDs in an array. If four FMDs are not enough, add a fifth captured and converted FMD to the array and send to CreateEnrollmentFMD(), and so on until an FMD is created.

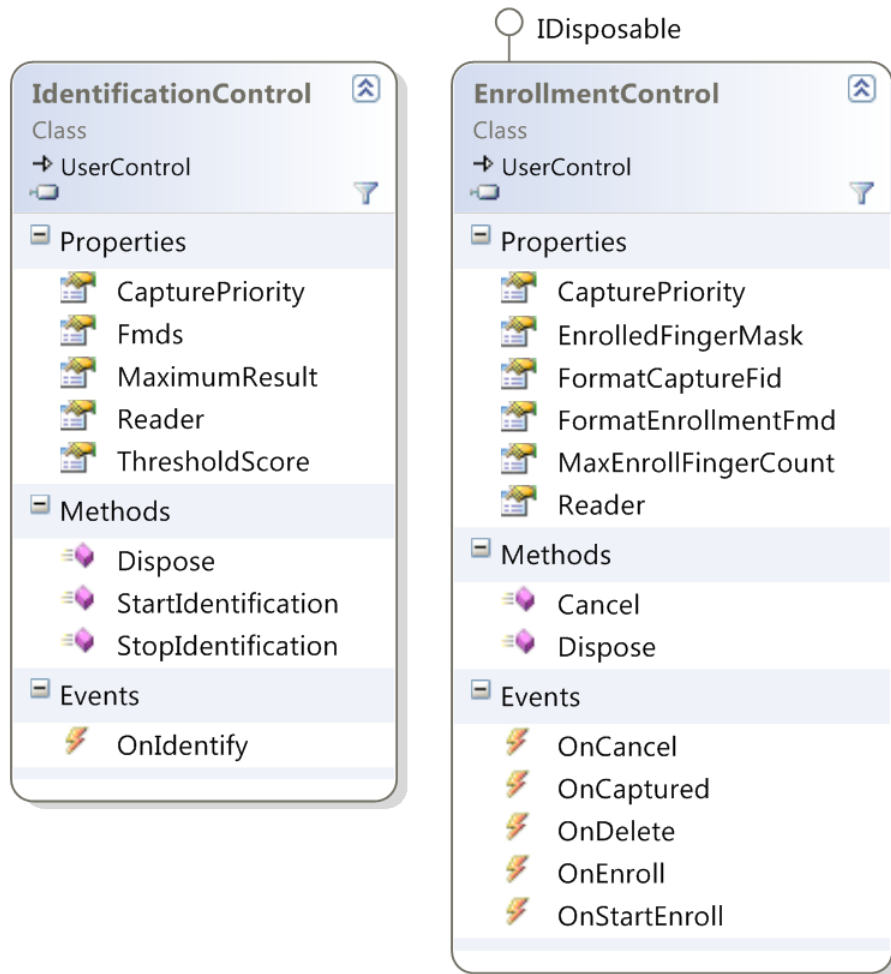
### 11.7.1 Common Data Structures for Results

The following classes are used when data is returned as a result of a Digital Persona .NET operation.



### 11.7.2 Pre-Built Controls for Enrollment and Identification

In addition to the basic API, the SDK contains two pre-built controls that allow you to quickly add enrollment or identification into your application. These controls provide the same functionality as the Digital Persona One Touch products.





## 12 The ActiveX API

---

This chapter provides an overview of the API. For details of using the API on Windows-based readers, consult the HID DigitalPersona Biometric SDK Platform Guide for Windows.

The ActiveX API and controls are built as a wrapper to the C/C++ APIs, and is available for the Windows platform.

Note that ActiveX is a Microsoft technology and is not supported on Mozilla Firefox and Google Chrome.

### 12.1 Importing the Digital Persona ActiveX package

The Digital Persona ActiveX library classes are aggregated into two DLLs:

- DPXUru.dll – ActiveX API library
- DPCTIXUru.dll – ActiveX GUI controls

### 12.2 Getting Detailed Documentation

This chapter provides an overview of the main methods in the ActiveX API. For a complete description of method parameters, the Doxygen documentation for the ActiveX libraries is provided. Consult the platform guide for details of where the Doxygen files are located.

### 12.3 Using the Package

#### 12.3.1 Main Access Points

The main access point to the Digital Persona ActiveX library is the XReaderCollection class.

- To acquire a reference to XReaderCollection use the GetReaders() method. To destroy XReaderCollection, (release all system resources associated with readers and make readers available for other processes) use the Dispose() method.
- To acquire a reference(s) to individual readers, use the XReaderCollection object, which is a collection of objects of type XReader.
- To work with the FingerJet Engine, instantiate one of the classes: XComparison, XImporter, XFeatureExtraction or XEnrollment. For example, to start enrolling fingerprints using the FingerJet engine, call the static method, Enrollment.CreateEnrollmentFmd(). This method takes as input an enumeration that specifies the format the enrollment template should be in, e.g., ANSI or ISO. This method also takes an IEnumerable<Fmd> object. See [Section 11.3.4 IEnumerables in the .NET Wrapper](#) for more information on IEnumerable<Fmd>.

### 12.3.2 SDKException

The SDKException class describes exceptions specific to the DigitalPersona Biometric SDK.

### 12.3.3 Serialization

The ActiveX API provides the ability to convert an FMD or FID into a format that can be read serially. Serialization allows you to easily transmit and store data as byte strings, streams or XML. This allows you to transmit the data or save it to standard file systems. The XML format can be used within HTML for building browser-based applications.

To return the data to its original format, deserialization is also provided.

Note that serialized FMDs and FIDs include the version number of the ActiveX API that was used for the serialization. If you attempt to deserialize with an older version of the API, the results may not be correct, so when deserializing, the API will throw an SDKException if the serialization was done using a later version of the wrapper.

Serialization in the API occurs using the default System.Xml.Serialization.XmlSerializer. The name of the root element of the generated XML is the same as the object type. Each public member of the object is represented as an XML element. Raw byte data is encoded as a Base-64 string by XmlSerializer.

Deserialization also uses XmlSerializer. The XML that was serialized is fed into XmlSerializer where the elements with values become object members with values of the same name as the element.

## 12.4 Working with Readers

Readers are accessed through the XReaderCollection object, which is a collection of XReader objects. Each attached reader is represented with a XReader object. The XReader interface allows:

- Querying reader description and capabilities,
- Acquiring status of the reader,
- Capturing fingerprints,
- Starting and stopping image stream, and
- Resetting and calibrating the reader.

To acquire a reader, first instantiate a XReaderCollection object and call XReaderCollection.GetReaders() which returns the attached readers. Iterate through this list to look at the description object for the desired reader. Once finished with all of the XReader objects in the XReaderCollection, ensure that you call the Dispose() method to destroy the XReaderCollection object and release system resources.

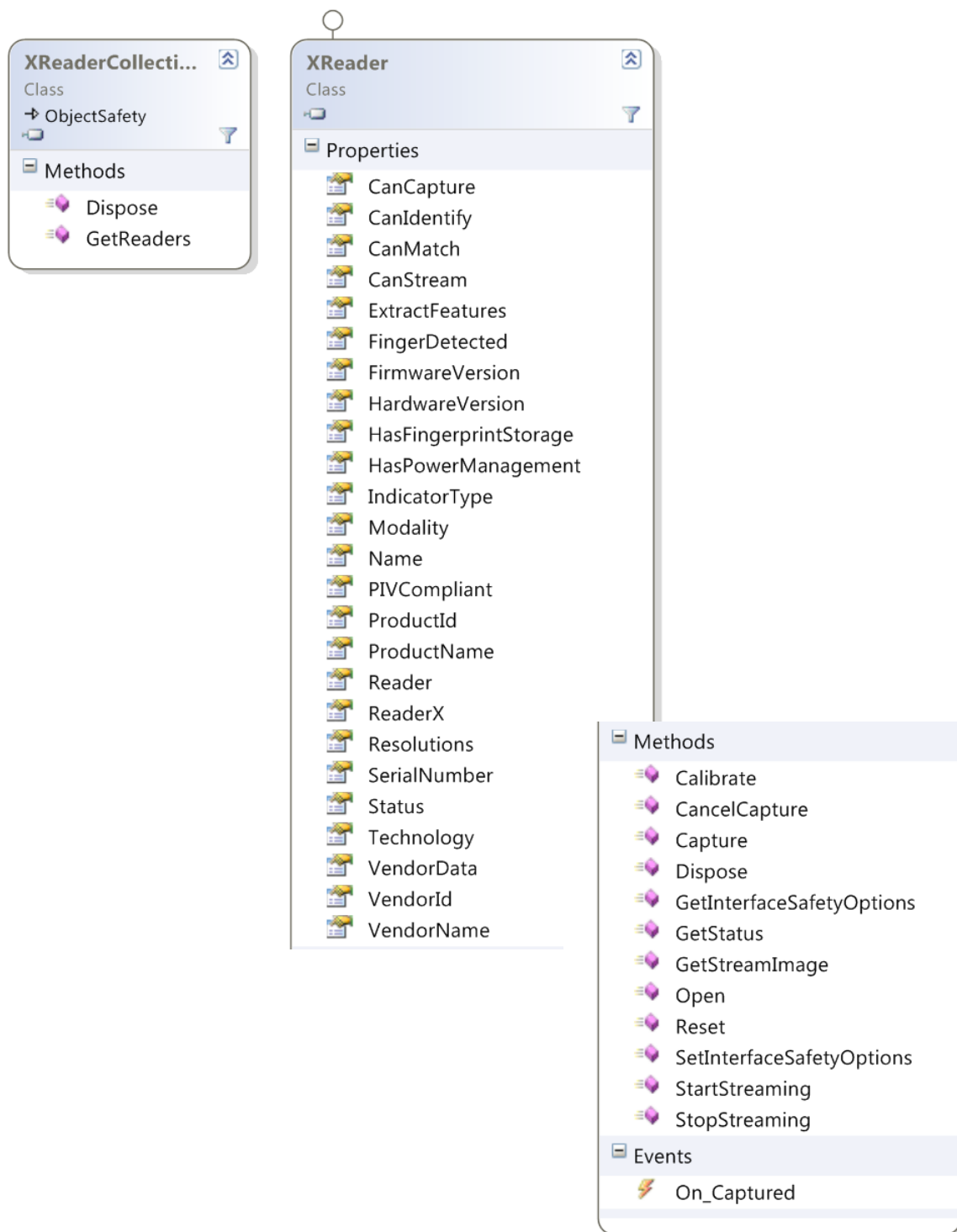
### 12.4.1 A Note About Internet Explorer and Process Merging

By default, Internet Explorer merges browser processes where it can, beginning with IE8. If your application is open in more than one browser window, users may experience erroneous behavior when your application calls XReader.Dispose() from one window.

To prevent this, users can use the File > New Session command to open a new window that does not merge processes with existing windows. You can also prevent IE from merging processes by using the -nomerge command line option when launching IE OR change the registry setting of HKEY\_CURRENT\_USER\Software\Microsoft\Internet Explorer\Main to set SessionMerging to 0. Note however that these options will affect Internet Explorer/s efficiency and startup time.

## 12.4.2 Class Diagrams

The UML diagrams below show the XReaderCollection and XReader classes.



The main reader hardware management methods in the XReader class are:

Function	Description
Open	Open a device and return the device capabilities. This method establishes an exclusive link to the device; no other processes will be able to use the device until you close it. The application <i>must</i> open the device before use.
GetStatus	Get the status for a device. You would normally check the device status before captures to ensure that the device is functioning and there are no error conditions.  Returns an object of type <code>ReaderStatus</code> which describes the current status of the reader.
Calibrate	Calibrate a device. Some devices are self-calibrating. Ambient light or temperature can affect calibration, for some devices. Calibration can take several seconds.
Reset	Do a hardware reset on the reader. Hardware resets are typically needed only after a hardware problem (e.g., the device is unplugged or receives an electrostatic shock). Hardware resets typically only take a few milliseconds.
Dispose	Close device, release memory, remove child objects.

The Capture function of the XReader class captures a fingerprint image for

- Enrollment (as part of the process described in [Section 3.3 Workflow - Enrollment Application.](#))
- Identifying users with Identify
- Verifying a specific user identity with Compare

The primary fingerprint capture methods are:

Function	Description
Capture	Captures a fingerprint image from the open reader. This function signals the reader that a fingerprint is expected, and blocks until a fingerprint is received, capture fails or the reader times out.
CancelCapture	Cancel a pending capture

## Streaming Fingerprints

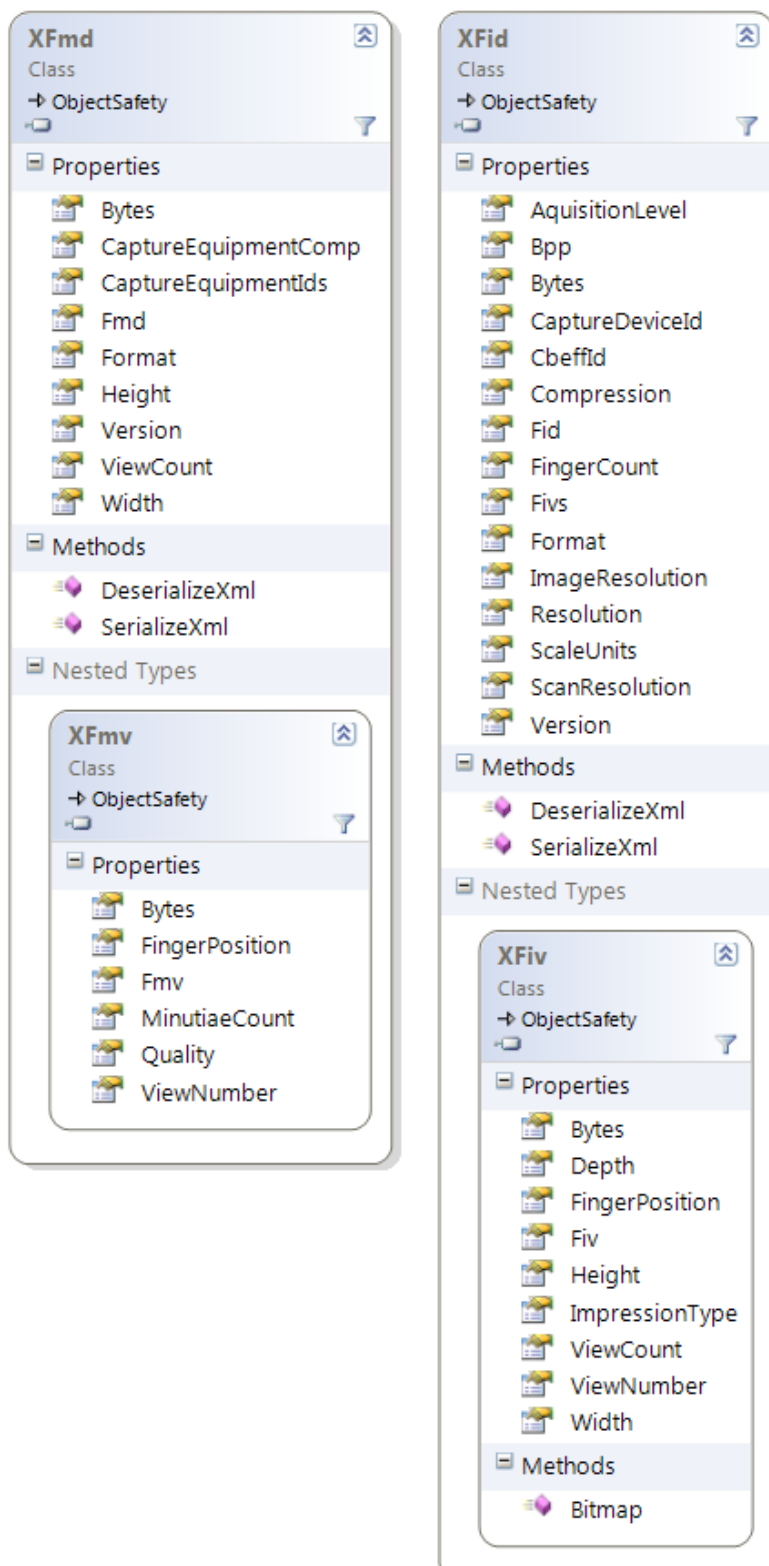
Not all readers support streaming mode. To determine if a specific reader supports this feature, check the value of the XReader property `CanStream`.

The streaming methods in XReader are:

Function	Description
StartStreaming	Put the reader into streaming mode. In this mode, the application must call <code>GetStreamImage()</code> to acquire images from the stream.
GetStreamImage	Capture a fingerprint image from the streaming data.  After this function returns, the reader remains in streaming mode.  Frame selection, scoring and other image processing are not performed by this function.
StopStreaming	End streaming mode

## 12.5 Managing Fingerprint Data

The ActiveX API implements fingerprint data as shown in the UML diagram below:



In addition to the usual methods for working with data, this class includes the methods below for serializing and deserializing (as described in [Section 11.1 Importing the Digital Persona .NET package](#)).

Function	Description
SerializeXml	Static method which serializes FMDs and FIDs into an XML string. Note that raw data is encoded as base-64.
DeserializeXml	Static method which de-serializes XML-encoded FMDs and FIDs back into a Digital Persona FMD or FID data structure.

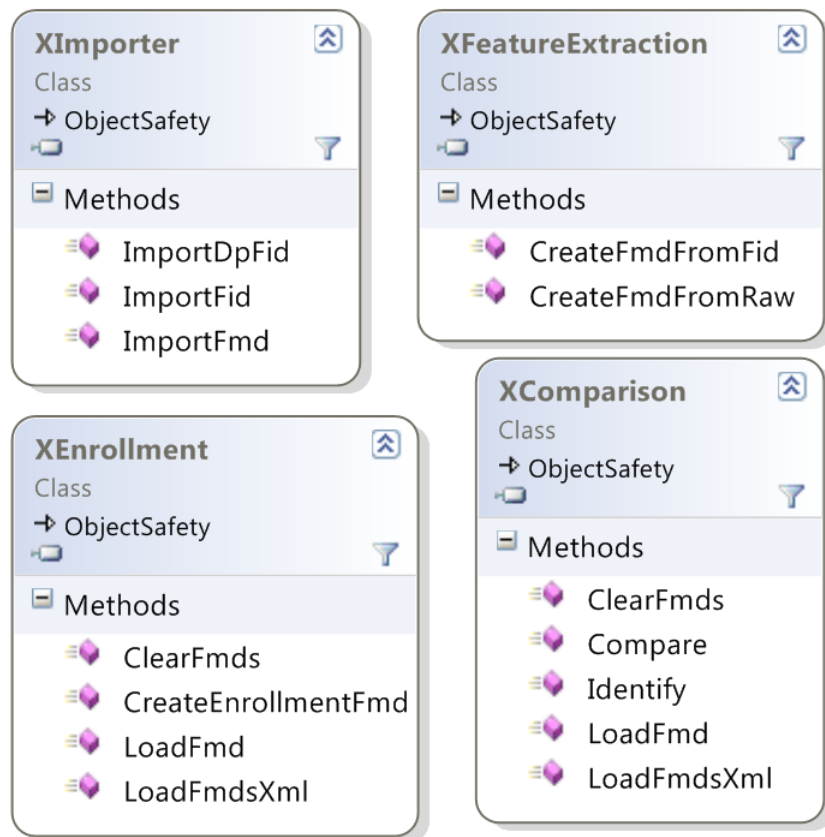
## 12.6 Accessing the FingerJet Engine

The FingerJet Engine interfaces provide functionality to

- Extract fingerprint features (create FMDs),
- Identify and compare FMDs, and
- Create enrollment FMDs.

Refer to [Section 3.2 Understanding the Data Flow](#) for a description of enrollment and comparison terminology and data flow.

The UML diagram below shows the relevant classes:



## 12.6.1 Creating FMDs from images

The `CreateFmdFromFid()` method can be used in two ways:

1. Extract fingerprint minutiae from a raw image and create an FMD.
2. Extract fingerprint minutiae from an FID and create an FMD.

The following limitations are applied to the raw images and FIDs:

- 8 bits per pixel
- no padding
- square pixels (horizontal and vertical dpi are the same)

The size of the resulting FMD will vary depending on the minutiae in a specific fingerprint.

## 12.6.2 Identification and Comparison

`Identify()` identifies a single FMD against an array of FMDs. This function takes as inputs:

- A single view in an FMD
- An array of FMDs (each FMD can contain up to 16 views) to compare
- The desired number of candidates to return
- The threshold for False Positive Identification Rate (FPIR) that is permitted

and returns matches as an array of integer pairs which provide the finger index and view index of each match.

Each time a view has a score lower than the threshold FPIR, that view is marked as a possible candidate. Then when all possible candidates are identified (i.e., they meet the threshold), they are ranked by their score. Finally, the function returns as many candidates as requested, based on the candidates with the lowest dissimilarity score. For a discussion of setting the threshold as well as the statistical validity of the dissimilarity score and error rates, consult [Section 3.5.3 NIST Fingerprint Image Quality \(NFIQ\)](#).

`Compare()` takes two single views from two FMDs and returns a dissimilarity score indicating the quality of the match.

The majority of applications should use the `Identify` method to implement both identification and verification. However, in a few special cases, e.g., using multi-modal biometrics, or doing statistical risk assessment, the `Compare` method allows you to compare two FMVs to determine their actual degree of dissimilarity. This is useful for accuracy testing and diagnostics and is not intended to be used in final applications for actual fingerprint recognition. The `Compare` method returns a dissimilarity score with values:

- 0 = fingerprints are NOT dissimilar (i.e., they MATCH perfectly).
- `maxint` (`#7FFFFFFF` or `2147483647`) = fingerprints are completely dissimilar (i.e., DO NOT match).
- Values close to 0 indicate very close matches, values closer to `maxint` indicate very poor matches.

The table below shows the relationship between the scores returned from `Compare` and the false match error rates observed in our test. The dissimilarity score distribution is estimated based on our internal testing, and may not be representative of the actual rate that will be observed in deployment.

Dissimilarity Score	False Match Rate
2147483	.1%
214748	.01%
21474	.001%
2147	.0001%

Function	Description
Compare	Compare two FMDs; supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.
Identify	Identify an FMD: given an array of FMDs, this function returns an array of candidates that match the original fingerprint (an FMV within an FMD) within the threshold of error. Supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.

### 12.6.3 Enrollment

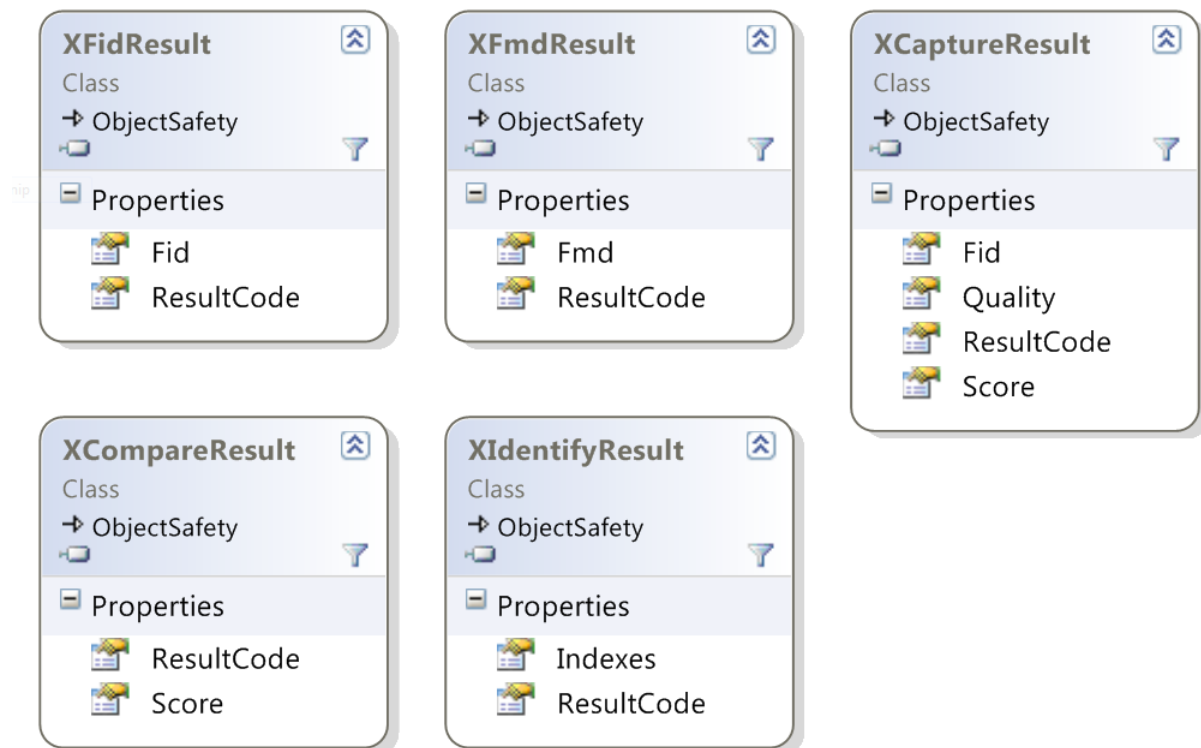
CreateEnrollmentFmd() creates and returns an enrollment FMD. The method takes as input:

- Which format to use, e.g., ANSI or ISO
- An ArrayList object containing the FMDs to enroll

The recommended sequence in which to enroll FMDs is to send CreateEnrollmentFmd() four captured FMDs in an array. If four FMDs are not enough, you can capture an FID, convert it to an additional FMD, add it to the array and call CreateEnrollmentFMD() again. If CreateEnrollmentFMD(), fails again, you can continue to add FMDs to the array and call CreateEnrollmentFMD() until an enrollment FMD is created successfully.

### 12.6.4 Common Data Structures for Results

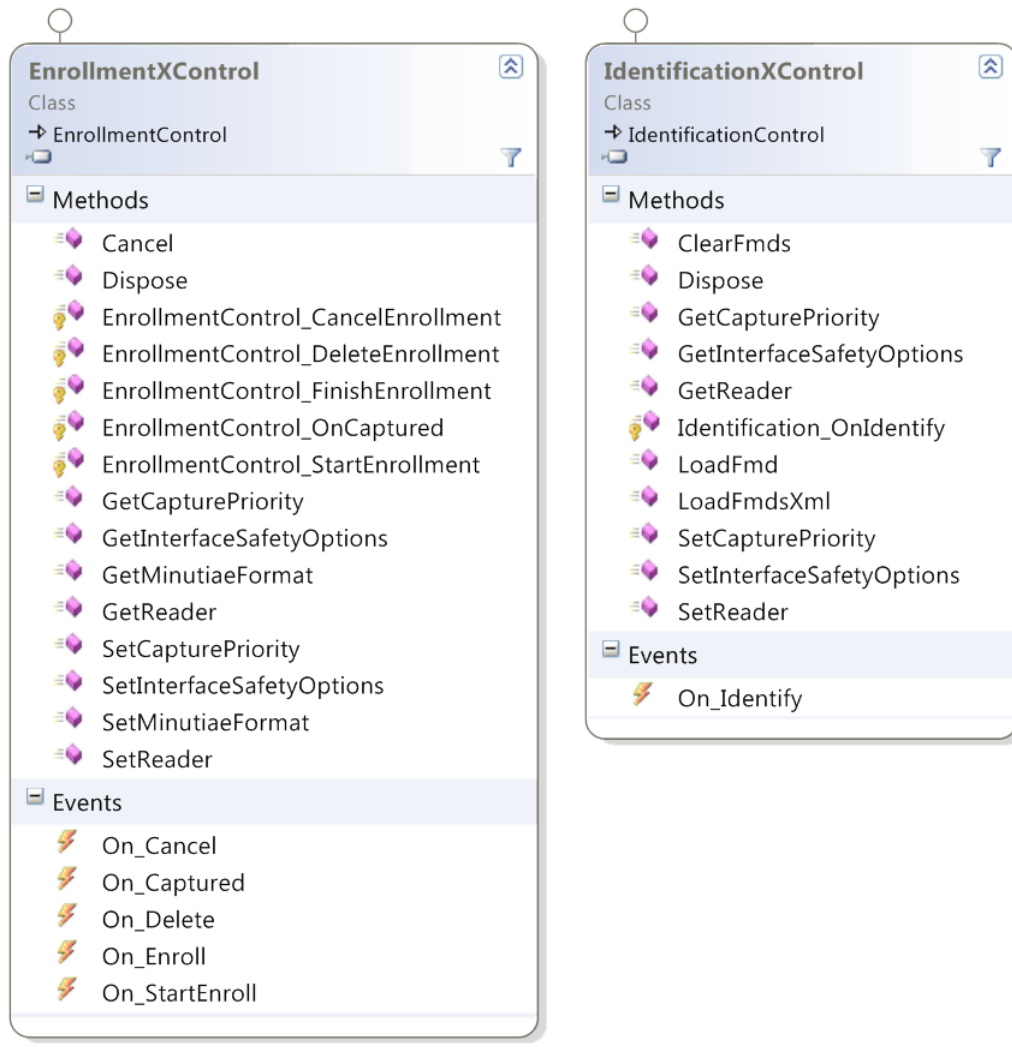
The following classes are used when data is returned as a result of a Digital Persona ActiveX operation.





### 12.6.5 Pre-Built Controls for Enrollment and Identification

In addition to the basic API, the SDK contains two pre-built controls that allow you to quickly add enrollment or identification into your application. These controls provide the same functionality as the Digital Persona One Touch products.





## 13 The JavaPOS API

---

The Digital Persona JavaPOS API is built on the Digital Persona Java API. This JavaPOS API is geared for Point of Sale applications and has the following features:

- Fully compliant with JavaPOS 1.13. The Digital Persona JavaPOS API conforms to the specifications for the Biometrics device category in Chapter 5, “Biometrics,” of the UnifiedPOS Retail Peripheral Architecture, Version 1.13 (July 15, 2009).
- The complete UPOS documentation is available at <http://www.nrf-arts.org/UnifiedPOS/default.htm>.
- Backward compatible with previous Digital Persona JavaPOS product (Digital Persona UPOS for JavaPOS SDK), with only a few caveats. This new API is the result of merging the previous JavaPOS SDK with the DigitalPersona Biometric SDK: the internal architecture has been completely rewritten. For users of the previous SDK, the updated SDK means an upgraded internal architecture for more robust performance -- up to ten times faster for identification. If you are upgrading an existing application, see [Section 5.6 Upgrading from Digital Persona UPOS for OPOS/JavaPOS](#) for more details.
- In addition to the JavaPOS API, the DigitalPersona Biometric SDK includes a JavaPOS Device Service object, which can be used with any JavaPOS Device Control for the Biometrics device category.
- Because the JavaPOS API is built as an adjunct to the Java API, applications can use both the JavaPOS standard operations AND use the Digital Persona Java API (described in [Section 8 The Java API](#)). Java methods can be used to access streaming features or to import data.

### 13.1 Terminology Note

The Digital Persona JavaPOS API conforms to the standard terminology used by the 2009 JavaPOS specification, whereas the other APIs in the DigitalPersona Biometric SDK generally use terminology that matches evolving industry standards. If you are going to use the Digital Persona Java API along with JavaPOS, you need to note that in JavaPOS, the extracted fingerprint data (template) is stored in a biometric information record (BIR). The templates created through enrollment and capture are equivalent to Fingerprint Minutiae Data (FMD) records in the other APIs of the DigitalPersona Biometric SDK. Note that JavaPOS templates include a 10- or 45-byte JavaPOS header in addition to the data itself.

### 13.2 Working with Fingerprint Data in JavaPOS

The Digital Persona JavaPOS API provides two types of fingerprint data: raw images and fingerprint templates. The data flow for JavaPOS applications is the same as described in [Section 3.2 Understanding the Data Flow](#), except that JavaPOS data has additional JavaPOS headers and JavaPOS does not use the FID/FMD terminology used by Digital Persona. Note that there is no easy method provided in the API for converting data from JavaPOS BIRs to FMDs.

### 13.2.1 Fingerprint Data for Raw Images (Captures)

Per the JavaPOS specification, raw fingerprint scans (images) are available in the RawSensorData property when StatusUpdateEvent triggers.

### 13.2.2 Fingerprint Data for Captures and Enrollment Templates (BIRs)

The Digital Persona JavaPOS API now supports four different JavaPOS biometric information record (BIR) formats, as set by the Algorithm property:

Function	Description
0	Default (Same as 1)
1	Digital Persona format with 45-byte header that conforms to JavaPOS 1.13 spec. New captures and enrollments will be done in the Digital Persona format with 45-byte headers. Verify and identify will correctly compare with BIRs that have 45-byte headers, the previous 10-byte headers or a mixture.
2	ANSI INSITS 378-2004. New captures and enrollments will be created in ANSI format with a 45-byte JavaPOS header. Verify and identify will require that all BIRs be in ANSI format.
3	ISO/IEC 19794-2:2005. New captures and enrollments will be created in ISO format with a 45-byte JavaPOS header. Verify and identify will require that all BIRs be in ISO format.
4	JavaPOS 1.x compatible. This is the previous Digital Persona format with 10-byte header. This format was the default (and only) format used in the Digital Persona UPOS for JavaPOS SDK. New captures and enrollments will be created in the Digital Persona format with 10-byte headers. Verify and identify will correctly compare with BIRs that have 10-byte headers.

**Identification and verification require that all BIRs be in the same format.**

Stored enrollment BIRs can be in one of four formats:

- Digital Persona format (10-byte headers, 45-byte headers or a mixture),
- ANSI (with 45-byte headers)
- ISO (with 45-byte headers)
- JavaPOS 1.x compatible (Digital Persona format with 10-byte headers).

For example, if the Algorithm property is set to ANSI and you receive ANSI captures, those fingerprints cannot be matched against templates that are stored in a Digital Persona format.

The value of the Algorithm property must be set before the device is enabled (i.e., after the device is opened and claimed, but before it is enabled).

### 13.2.3 Working with Digital Persona Record Formats

The format for the new default 45-byte headers is provided in the JavaPOS specifications. The first 10 bytes of the 45-byte header are the same as the previous 10-byte header except for the header version number.

In both 10-byte and 45-byte headers, the fifth byte specifies the header format:

- The fifth byte is 0x10 for a 10-byte header
- The fifth byte is 0x20 for a 45-byte header

### 13.2.4 Converting to a Different Data Format

If you wish to convert to a different data format, you must re-enroll users with the new format. There is no way to convert data between formats.

## 13.3 Getting Device-Specific Information with DirectIOEvent

The DirectIOEvent event is fired by a Service Object to deliver vendor-specific events to the application. The Digital Persona JavaPOS API uses DirectIOEvent events to notify the application about device connection and disconnection and intermediate events such as finger touch or removal.

### Syntax

```
upos::events::DirectIOEvent
  EventNumber: int32 { read-only }
  Data: int32 { read-write }
  Obj: object { read-write }
```

### Properties

This event contains the following attributes:

Function	Type	Description
<i>EventNumber</i>	<i>int</i>	Constants whose specific values are assigned by the Service Object
<i>Data</i>	<i>int</i>	Not used
<i>Obj</i>	<i>object</i>	Name of the fingerprint reader

### EventNumber Return Values

These constants are defined in the `com.Crossmatch.javapos.services.biometrics.dpfconstants` class.

Event Number	Description
DP_EVENT_DISCONNECT	The fingerprint reader was disconnected.
DP_EVENT_RECONNECT	The fingerprint reader was reconnected.
DP_EVENT_FINGER_TOUCHED	The fingerprint reader was touched.
DP_EVENT_FINGER_GONE	The finger was removed from the fingerprint reader.
DP_EVENT_COMPLETED	Capture completed.
DP_EVENT_IMAGE_READY	Raw image (from a capture) is available.
DP_EVENT_SAMPLE_QUALITY	Information about quality of the sample is available.

## 13.4 Implementation Notes

The following table provides information about how UPOS and JavaPOS properties, methods, and events are implemented in the Digital Persona JavaPOS API.

Property	Type	Description
UPOS Common Properties		
AutoDisable	No	This property is initialized to false in the open method. Changing its value will have no effect.
CapCompareFirmwareVersion	Yes	This property is initialized to false in the open method since firmware comparison is not supported.
CapPowerReporting	Yes	This property is initialized to JPOS_PR_NONE in the open method since power reporting is not supported.
CapStatisticsReporting	Yes	This property is initialized to false in the open method since statistics reporting is not supported.
CapUpdateFirmware	Yes	This property is initialized to false in the open method since firmware updating is not supported.
CapUpdateStatistics	Yes	This property is initialized to false in the open method since statistics updating is not supported.
CheckHealthText	Yes	This property is initialized to an empty string in the open method. It is never modified since device health checking is not supported.
Claimed	Yes	
DataCount	Yes	
DataEventEnabled	Yes	Has value true when an enroll or capture or verify capture is in progress.
DeviceEnabled	Yes	Has value true when the service will return device events. The Algorithm property cannot be changed when the device is enabled.
DeviceServiceDescription	Yes	This property is set to "Digital Persona Biometrics Device Service for JavaPOS, (c) 2007-2015 Crossmatch."
DeviceServiceVersion	Yes	This property is set to 1013000.
FreezeEvents	Yes	
PhysicalDeviceDescription	Yes	This property contains the product name, serial number, and vendor name for the Digital Persona device.
PhysicalDeviceName	Yes	This property contains the product name of the Digital Persona device.
PowerNotify	Yes	This property is initialized to JPOS_PN_DISABLED in the open method since power reporting is not supported.
PowerState	Yes	This property is initialized to JPOS_PS_UNKNOWN in the open method since power reporting is not supported.
State	Yes	
Specific Properties		
Algorithm	Yes	Specifies the data format used. See <a href="#">Section 13.2.2 Fingerprint Data for Captures and Enrollment Templates (BIRs)</a> for details.

Property	Type	Description
AlgorithmList	Yes	See <a href="#">Section 13.2.2 Fingerprint Data for Captures and Enrollment Templates (BIRs)</a> for details.
BIR	Yes	
CapPrematchData	Yes	This property is initialized to false in the open method since pre-matching is not supported.
CapRawSensorData	Yes	
CapRealTimeData	Yes	This property is initialized to false in the open method.
CapSensorColor	Yes	This property is initialized to BIO_CSC_GRAYSCALE in the open method.
CapSensorOrientation	Yes	This property is initialized to BIO_CSO_NORMAL in the open method.
CapSensorType	Yes	This property is initialized to BIO_CST_FINGERPRINT in the open method.
CapTemplateAdaptation	Yes	This property is initialized to false in the open method since template adaptation is not supported.
RawSensorData	Yes	Unknown OR contains raw sensor data. The data in this property exists when CapRawSensorData is set to true. and StatusUpdateEvent triggers.
RealTimeDataEnabled	Yes	This property is initialized to false in the open.
SensorBPP	Yes	This property is initialized to 8 in the open method. No other value is supported.
SensorColor	Yes	This property is initialized to BIO_CSC_GRAYSCALE in the open method. No other value is supported.
SensorHeight	Yes	This property is initialized to 0 in the open method. No other value is supported.
SensorOrientation	Yes	This property is initialized to BIO_SO_NORMAL in the open method. No other value is supported.
SensorType	Yes	This property is initialized to BIO_ST_FINGERPRINT in the open method.
SensorWidth	Yes	This property is initialized to 0 in the open method. No other value is supported.
UPOS Common Methods		
Open	Yes	
Close	Yes	
ClaimDevice	Yes	
ReleaseDevice	Yes	
CheckHealth	Yes	Only internal health check is supported. Attempts to perform an external or interactive check cause an exception to be thrown.
ClearInput	Yes	
ClearInputProperties	Yes	
ClearOutput	No	This method is not supported by the Biometrics device category.

Property	Type	Description
DirectIO	Partial	An exception is thrown since device direct I/O is not supported.
CompareFirmwareVersion	Yes	An exception is thrown since CapCompareFirmwareVersion is set to false.
ResetStatistics	Yes	An exception is thrown since CapUpdateStatistics is set to false.
RetrieveStatistics	Yes	An exception is thrown since CapStatisticsReporting is set to false.
UpdateFirmware	Yes	An exception is thrown since CapUpdateFirmware is set to false.
UpdateStatistics	Yes	An exception is thrown since CapUpdateStatistics is set to false.
Specific Methods		
beginEnrollCapture	Yes	
beginVerifyCapture	Yes	
endCapture	Yes	
identify	Yes	
identifyMatch	Yes	
processPrematchData	Yes	An exception is thrown since CapPrematchData is set to false.
verify	Yes	
verifyMatch	Yes	
UPOS Events		
DataEvent	Yes	The following event types are supported: BIO_DATA_ENROLL - Enrollment template created BIO_DATA_VERIFY - Verification template created
DirectIOEvent	Yes	See <a href="#">Section 13.3 Getting Device-Specific Information with DirectIOEvent</a> for details.
ErrorEvent	Yes	resultCode - Standard JavaPOS Result Code resultCodeExtended - Digital Persona Result Code - see <i>Section 13.6 Device-Related Error Codes</i> for details errorLocus - EL_INPUT errorResponse - ER_CLEAR
OutputCompleteEvent	No	This event is not supported by the Biometrics device category.
StatusUpdateEvent	Yes	Only occurs when RawSensorData has received a raw image from a capture.

## 13.5 Exceptions

The files `jpos.Const.*` contain the JavaPOS standard exception codes. The following exceptions are thrown by the Digital Persona JavaPOS API:

- JPOS\_E\_FAILURE
- JPOS\_E\_ILLEGAL
- JPOS\_E\_NOHARDWARE



- JPOS\_E\_TIMEOUT

## 13.6 Device-Related Error Codes

For the device-related result codes returned by `EventNumber` after the `DirectIOEvent` event, see *EventNumber* **Return Values** [on page 85](#).

The following error codes are returned in the `ResultCodeExtended` property of the `ErrorEvent` object.

**IMPORTANT:** You should always use the names of the return codes, such as `FT_OK`, rather than the numeric values because these values may change at any time, without notice. The numbers are provided as a reference for the error codes in the sample application.

Property	Type	Description
-1	FT_ERR_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module is not initialized.
-2	FT_ERR_INVALID_PARAM	One or more parameters are not valid.
-4	FT_ERR_IO	A generic I/O file error occurred.
-7	FT_ERR_NO_MEMORY	There is not enough memory to perform the action.
-8	FT_ERR_INTERNAL	An unknown internal error occurred.
-10	FT_ERR_UNKNOWN_DEVICE	The requested device is not known.
-11	FT_ERR_INVALID_BUFFER	A buffer is not valid.
-33	FT_ERR_UNKNOWN_EXCEPTION	An unknown exception occurred.



## 14 The OPOS API

---

The Digital Persona OPOS API is built on the Digital Persona C/C++ API. The OPOS API is geared for Point of Sale applications and has the following features:

- Fully compliant with OPOS 1.13. The Digital Persona OPOS API conforms to the specifications for the Biometrics device category in Chapter 5, “Biometrics,” of the UnifiedPOS Retail Peripheral Architecture, Version 1.13 (July 15, 2009). The complete UPOS documentation is available at <http://www.nrf-arts.org/UnifiedPOS/default.htm>.
- Backward compatible with previous Digital Persona OPOS product (Digital Persona UPOS for OPOS SDK), with a few minor caveats. This new API is the result of merging the previous OPOS SDK with the DigitalPersona Biometric SDK. For users of the previous SDK, the updated SDK means an upgraded internal architecture for more robust performance -- up to ten times faster for identification. If you are upgrading an existing application, see [Section 5.6 Upgrading from Digital Persona UPOS for OPOS/JavaPOS](#) for more details.
- The OPOS API is a set of COM objects that acts as an extension to the DigitalPersona fingerprint reader driver, providing an OPOS-compliant application interface to Digital Persona products. It consists of a Control Object (CO) (an ActiveX® Control) for the OPOS Biometrics device category and a Service Object (SO).
- Because the OPOS API is built as an adjunct to the C/C++ API, applications can use both the OPOS standard operations AND use the Digital Persona C/C++ API (described in [Section 7 The C++ APIs](#)). C/C++ methods can be used to access streaming features or to import data.
- A complete reference to the controls, properties, methods and events of the OPOS interface is provided in Appendix A of the UnifiedPOS specification, a PDF document titled “UnifiedPOS Retail Peripheral Architecture.” Download the current version from <http://www.nrf-arts.org/content/unifiedpos>.

### 14.1 Terminology Note

The Digital Persona OPOS API conforms to the standard terminology used by the 2009 OPOS specification, whereas the other APIs in the DigitalPersona Biometric SDK generally use terminology that matches evolving industry standards. If you are going to use the Digital Persona C/C++ API along with OPOS, you need to note that in OPOS, the extracted fingerprint data (template) is stored in a biometric information record (BIR). The templates created through enrollment and capture are equivalent to Fingerprint Minutiae Data (FMD) records in the other APIs of the DigitalPersona Biometric SDK. Note that OPOS templates include a 12- or 45-byte OPOS header in addition to the data itself.

### 14.2 Working with Fingerprint Data in OPOS

The Digital Persona OPOS API provides two types of fingerprint data: raw images and fingerprint templates. The data flow for OPOS applications is the same as described in [Section 3.2 Understanding the Data Flow](#), except that OPOS data has additional OPOS headers and OPOS does not use the FID/FMD terminology used

by Digital Persona. Note that there is no easy method provided in the API for converting data from OPOS BIRs to FMDs.

### 14.2.1 Fingerprint Data for Raw Images (Captures)

Per the OPOS specifications, raw fingerprint scans (images) are available in the RawSensorData property when StatusUpdateEvent triggers.

### 14.2.2 Fingerprint Data for Captures and Enrollment Templates (BIRs)

The Digital Persona OPOS API now supports four different OPOS biometric information record (BIR) formats, as set by the Algorithm property:

Value	Meaning
0	Default (Same as 1)
1	Digital Persona format with 45-byte header that conforms to OPOS 1.13 spec. New captures and enrollments will be done in the Digital Persona format with 45-byte headers. Verify and identify will correctly compare with BIRs that have 45-byte headers, the previous 10-byte headers or a mixture.
2	ANSI INSITS 378-2004. New captures and enrollments will be created in ANSI format with a 45-byte OPOS header. Verify and identify will require that all BIRs be in ANSI format.
3	ISO/IEC 19794-2:2005. New captures and enrollments will be created in ISO format with a 45-byte OPOS header. Verify and identify will require that all BIRs be in ISO format.
4	OPOS 1.x compatible. This is the previous Digital Persona format with 12-byte header. This format was the default (and only) format used in the Digital Persona UPOS for OPOS SDK. New captures and enrollments will be created in the Digital Persona format with 12-byte headers. Verify and identify will correctly compare with BIRs that have 12-byte headers.

**Identification and verification require that all BIRs be in the same format.**

The BIRs can be in any of four formats:

- Digital Persona format (12-byte headers, 45-byte headers or a mixture),
- ANSI (with 45-byte headers)
- ISO (with 45-byte headers)
- OPOS 1.x compatible (Digital Persona format with 12-byte headers).

For example, if the Algorithm property is set to ANSI and you receive ANSI captures, those fingerprints cannot be matched against templates that are stored in a Digital Persona format.

The value of the Algorithm property must be set before the device is enabled (i.e., after the device is opened and claimed, but before it is enabled).

### 14.2.3 Working with Digital Persona Record Formats

The format for the new default 45-byte headers is provided in the OPOS specifications. The first 12 bytes of the 45-byte header are the same as the previous 12-byte header except for the header version number.

In both 12-byte and 45-byte headers, the 5th byte specifies the header format:

- The 5th byte is 0x10 for a 12-byte header
- The 5th byte is 0x20 for a 45-byte header

## 14.2.4 Converting to a Different Data Format

If you wish to convert to a different data format, you must re-enroll users with the new format. There is no way to convert data between formats.

## 14.3 Getting Device-Specific Information with DirectIOEvent

The DirectIOEvent event is fired by a Service Object to deliver vendor-specific events to the application. The Digital Persona JavaPOS API uses DirectIOEvent events to notify the application about device connection and disconnection and intermediate events such as finger touch or removal.

This chapter contains specific information about the DigitalPersona Biometric SDK implementation of the OPOS Control.

### 14.3.0.1 DataEvent

#### Syntax

```
<< event >> upos::events::DataEvent
```

Status: int32 { read-only }

#### Description

This event is fired to provide input data from the fingerprint reader to the application. The actual input data is placed in one or more device-specific properties.

#### Attribute

This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	BIO_DATA_ENROLL if enroll capture is completed. BIO_DATA_VERIFY if verify capture is completed.

### 14.3.0.2 DirectIOEvent

#### Syntax

```
<< event >> upos::events::DirectIOEvent
  EventNumber: int32 { read-only }
  Data: int32 { read-write }
  Obj: object { read-write }
```

#### Description

This event is fired by the Service Object (SO) to communicate directly with the application. DirectIOEvent is used in the DigitalPersona Biometric SDK to notify the user of the image-capturing status, fingerprint reader connection status, and image quality, etc., whenever required.

#### Attributes

This event contains the following attributes:

Attribute	Type	Description
<i>EventNumber</i>	<i>int32</i>	Event number whose specific values are assigned by the SO.
<i>Data</i>	<i>int32</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the SO.
<i>Obj</i>	<i>object</i>	Additional data whose use varies by the <i>EventNumber</i> and the SO.

### EventNumber Return Values

EventNumber	Description
1	The fingerprint reader was disconnected.
2	The fingerprint reader was reconnected.
3	The fingerprint reader was touched.
4	The finger was removed from the fingerprint reader.
5	A fingerprint image is ready for processing.
6	Provides information about the quality of the fingerprint image.
7	A supplied fingerprint credential was added to the fingerprint enrollment template.
8	The fingerprint capture operation was stopped.

### Data Return Values

EventNumber*	Description
6	DP_QUALITY_GOOD, DP_QUALITY_NONE, etc.
7	1, 2, 3, etc.

\* For every other EventNumber (1 through 5 and 8), Data holds the value 0 (Not Supported).

### DP\_SAMPLE\_QUALITY Enumeration

Value	Meaning
DP_QUALITY_GOOD (0)	The image is of good quality.
DP_QUALITY_NONE (1)	There is no image.
DP_QUALITY_TOOLIGHT (2)	The image is too light.
DP_QUALITY_TOODARK (3)	The image is too dark.
DP_QUALITY_TOONOISY (4)	The image is too noisy.
DP_QUALITY_LOWCONTR (5)	The image contrast is too low.
DP_QUALITY_FTRNOTENOUGH (6)	The image does not contain enough information.
DP_QUALITY_NOCENTRAL (7)	The image is not centered.

### Obj Return Values

EventNumber	Object
1	The fingerprint reader was disconnected.
2	The fingerprint reader was reconnected.
3	The fingerprint reader was touched.
4	The finger was removed from the fingerprint reader.
5	NULL (Not Supported).

EventNumber	Object
6	Provides information about the quality of the fingerprint image.
7	A supplied fingerprint credential was added to the fingerprint enrollment template.
8	The fingerprint capture operation was stopped.

### 14.3.0.3 StatusUpdateEvent

#### Syntax

```
<< event >> upos::events::StatusUpdateEvent
    Status: int32 { read-only }
```

#### Description

This event is used in the DigitalPersona Biometric SDK to notify the user that a raw image is available for use.

#### Attribute

This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	The <i>Status</i> parameter notifies the user that raw image data is available.

#### Status Return Values

StatusUpdateEvent	Value	Meaning
1	BIO_SUE_RAW_DATA	Raw image data is available.

## 14.4 Implementation Notes

The following table provides information about how UPOS properties, methods, and events are implemented in the DigitalPersona Biometric SDK.

Name	Implemented*	Comment
AutoDisable	Partial	This property is initialized to false in the <b>open</b> method. It is not modified during execution of the application since it is not required to automatically disable the device when data is received.
BinaryConversion	Partial	This property is initialized to OPOS_BC_NONE in the <b>open</b> method and is not modified during execution of the application.
CapCompareFirmwareVersion	Partial	This property is initialized to false in the <b>open</b> method. It is not modified during execution of the application since firmware version comparison is not supported.
CapPowerReporting	Partial	This property is initialized to OPOS_PR_NONE in the <b>open</b> method. It is not modified during execution of the application since power reporting is not supported.
CapStatisticsReporting	Partial	This property is initialized to false in the <b>open</b> method. It is not modified during execution of the application since statistics reporting is not supported.

Name	Implemented*	Comment
CapUpdateFirmware	Partial	This property is initialized to false in the <b>open</b> method. It is not modified during execution of the application since firmware updating is not supported.
CapUpdateStatistics	Partial	This property is initialized to false in the <b>open</b> method. It is not modified during execution of the application since statistics updating is not supported.
CheckHealthText	No	
Claimed	Yes	This property is initialized to false in the <b>open</b> method. It is set to true on claim and set to false again on release.
ControlObjectDescription	Yes	
ControlObjectVersion	Yes	
DataCount	Partial	This property is initialized to 0 in the <b>open</b> method and is not modified during execution of the application.
DataEventEnabled	Partial	This property is initialized to false in the <b>open</b> method and is set to true after a successful claim operation.
DeviceDescription	Yes	
DeviceEnabled	Partial	This property is initialized to false in the <b>open</b> method and is set to true after a successful claim operation. Since the <b>AutoDisable</b> property is never set to true, <b>DeviceEnabled</b> always remains true.
DeviceEnabled	Partial	This property is initialized to false in the <b>open</b> method and is set to true after a successful claim operation. Since the <b>AutoDisable</b> property is never set to true, <b>DeviceEnabled</b> always remains true.
DeviceName	Yes	
FreezeEvents	Partial	This property is initialized to false in the <b>open</b> method. It is not modified during execution of the application since it is not required to freeze events.
OpenResult	Yes	
OutputID	No	This property is not supported by the Biometrics device category.
PowerNotify	Partial	This property is initialized to OPOS_PN_DISABLED in the <b>open</b> method. It is not modified during execution of the application since power reporting is not supported.
PowerState	Partial	This property is initialized to OPOS_PS_UNKNOWN in the <b>open</b> method. It is not modified during execution of the application since power reporting is not supported.
ResultCode	Yes	
ResultCodeExtended	No	
ServiceObjectDescription	Yes	
ServiceObjectVersion	Yes	
State	Partial	This property is initialized to OPOS_S_IDLE in the <b>open</b> method and is set to OPOS_S_CLOSED in the close method.



Name	Implemented*	Comment
Specific Properties		
Algorithm	Yes	See <a href="#">Section 14.2.2 Fingerprint Data for Captures and Enrollment Templates (BIRs)</a> for details.
AlgorithmList	Yes	See <a href="#">Section 14.2.2 Fingerprint Data for Captures and Enrollment Templates (BIRs)</a> for details.
BIR	Yes	This property holds the biometric data that is captured and returned to the application.
CapPrematchData	Partial	This property is initialized to false in the <b>open</b> method. It is not modified during execution of the application since the MOC (Match-on-Card) SmartCard technology needed to generate a processed BIR based on pre-match data stored on a SmartCard is not supported.
CapRawSensorData	Yes	This property is initialized to true in the <b>open</b> method and is not modified during execution of the application.
CapRealTimeData	Partial	This property is initialized to false in the <b>open</b> method and is not modified during execution of the application.
CapSensorColor	Yes	This property is initialized to BIO_CSC_GRAYSCALE in the <b>open</b> method and is not modified during execution of the application.
CapSensorOrientation	Yes	This property is initialized to BIO_CSO_INVERTED in the <b>open</b> method and is not modified during execution of the application.
CapSensorType	Yes	This property is initialized to BIO_CST_FINGERPRINT in the <b>open</b> method and is not modified during execution of the application.
CapTemplateAdaptation	Partial	This property is initialized to false in the <b>open</b> method. It is not modified during the execution of the application since template adaptation is not supported.
RawSensorData	Yes	This property holds the raw biometric data that is captured and returned to the application.
RealTimeDataEnabled	Partial	This property is initialized to false in the <b>open</b> method and is not modified during execution of the application.
SensorBPP	Yes	This property is initialized by invoking the <b>Digital Persona</b> function that defines the image resolution, in pixels.
SensorColor	Yes	This property is initialized by invoking the <b>Digital Persona</b> function that defines the image type.
SensorHeight	Yes	This property is initialized by invoking the <b>Digital Persona</b> function that defines the image height, in pixels.
SensorOrientation	Yes	This property is initialized to BIO_SO_INVERTED in the <b>open</b> method and is not modified during execution of the application.
SensorType	Yes	This property is initialized to BIO_ST_FINGERPRINT in the <b>open</b> method and is not modified during execution of the application.

Name	Implemented*	Comment
SensorWidth	Yes	This property is initialized by invoking the <b>Digital Persona</b> function that defines the image width, in pixels.
Common Methods		
Open	Yes	
Close	Yes	
ClaimDevice	Yes	
ReleaseDevice	Yes	
CheckHealth	No	
ClearInput	No	
ClearInputProperties	No	
ClearOutput	No	This method is not supported by the Biometrics device category.
DirectIO	No	
CompareFirmwareVersion	No	
ResetStatistics	No	
RetrieveStatistics	No	
UpdateFirmware	No	
UpdateStatistics	No	
Specific Methods		
beginEnrollCapture	Yes	
beginVerifyCapture	Yes	
endCapture	Yes	
identify	Yes	
identifyMatch	Yes	
processPrematchData	No	
verify	Yes	
verifyMatch	Yes	
Events		
DataEvent	Yes	Valid values for <i>Status</i> parameter are 1- BIO_DATA_ENROLL if enroll capture is completed. 2- BIO_DATA_VERIFY if verify capture is completed.

Name	Implemented*	Comment
DirectIOEvent	Specific	Return values are 1 - DP_DIOE_DISCONNECT 2 - DP_DIOE_RECONNECT 3 - DP_DIOE_FINGER_TOUCHED 4 - DP_DIOE_FINGER_GONE 5 - DP_DIOE_IMAGE_READY 6 - DP_DIOE_SAMPLE_QUALITY 7 - DP_DIOE_ENROLL_FEATURES_ADDED 8 - DP_DIOE_OPERATION_STOPPED
ErrorEvent	Yes	<b>ResultCode</b> - Standard OPOS Result Code
	Specific	<b>ResultCodeExtended</b> - Digital Persona Result Code
	Yes	<b>ErrorLocus</b> - EL_INPUT
	Yes	<b>ErrorResponse</b> - ER_CLEAR
OutputCompleteEvent	No	This event is not supported by the Biometrics device category.
StatusUpdateEvent	Partial	Valid values for <i>Status</i> parameter are 1 - BIO_SUE_RAW_DATA

\* Implemented: Yes = Fully implemented  
 No = Not implemented  
 Partial = Implemented but not all features are available  
 Specific = Implemented with device-specific features or data

## 14.5 Exceptions

The files `jpos.Const.*` contain the JavaPOS standard exception codes. The following exceptions are thrown by the Digital Persona JavaPOS API:

- JPOS\_E\_FAILURE
- JPOS\_E\_ILLEGAL
- JPOS\_E\_NOHARDWARE
- JPOS\_E\_TIMEOUT

## 14.6 Device-Related Error Codes

For the device-related result codes returned by `EventNumber` after the `DirectIOEvent` event, see [EventNumber Return Values on page 94](#).

The following error codes are returned in the `ResultCodeExtended` property of the `ErrorEvent` object.

IMPORTANT: You should always use the names of the return codes, such as `FT_OK`, rather than the numeric values because these values may change at any time, without notice. The numbers are provided as a reference for the error codes in the sample application.

Value	Result Code	Meaning
-1	FT_ERR_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module is not initialized.
-2	FT_ERR_INVALID_PARAM	One or more parameters are not valid.
-4	FT_ERR_IO	A generic I/O file error occurred.
-7	FT_ERR_NO_MEMORY	There is not enough memory to perform the action.
-8	FT_ERR_INTERNAL	An unknown internal error occurred.
-10	FT_ERR_UNKNOWN_DEVICE	The requested device is not known.
-11	FT_ERR_INVALID_BUFFER	A buffer is not valid.
-33	FT_ERR_UNKNOWN_EXCEPTION	An unknown exception occurred.
-1	FT_ERR_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module is not initialized.

## 15 The Objective-C API

The DigitalPersona Biometric SDK for iOS is an Objective-C based API for the fingerprint reader and includes support for iOS devices. It communicates with the fingerprint reader using the iOS External Accessories Framework. Detailed documentation for the API is contained in the header files. The API is thread-safe.

The SDK includes a sample Objective-C class called Reader inside the sample program UareUSample. The Reader class exposes the following methods.

### 15.1 DP Capture API

The DP Capture API consists of library management, reader management, capturing and streaming.

#### 15.1.1 Fingerprint Capture Device Management

Methods	Description
enumerateDevices()	The enumerateDevices() method queries the connected ExternalAccessories and returns zero if the fingerprint reader is plugged into the iOS device.
open()	Opens the session for communicating with the fingerprint reader.
getCapabilities()	Gets the capabilities of a device. Capabilities are returned as an array using the delegate method on the ReaderDelegate protocol.
close()	Closes a reader and releases the resources associated with the reader.

#### 15.1.2 Capturing Fingerprints

Methods	Description
capture()	Captures a fingerprint image from the open reader. This method signals the reader that a fingerprint is expected, and blocks until a fingerprint is received, the capture fails or the reader times out.
captureAsync()	Starts asynchronous capture on the opened reader device, signaling that a fingerprint is expected and then exits.
cancelCapture()	Cancels a pending capture.
getImageSize()	Returns the size of the raw image buffer. You can use this method, for example, to save the raw image data to a file.
getImageData()	Returns the raw bytes of the image buffer. You can use this method, for example, to save the raw image data to the file.

### 15.1.3 ReaderDelegate Methods

Methods	Description
didReceiveImage()	When the fingerprint reader is in streaming mode, the Reader gets the raw image buffer. It does the Core Graphics conversion of the raw image to UIImage, so it can be displayed on the iOS App UI. It calls this delegate method to send the UIImage to the registered delegate.
didReceiveImageData()	When the fingerprint reader is in streaming mode, the Reader gets the raw image buffer. This delegate method returns the raw image buffer without any CoreGraphics manipulation. Typically this is data used to write to file.
errorMsg()	If any error happens inside the Reader, this delegate method is used to communicate the error.
didCompleteCaptureAsync()	This is generated upon completion of an asynchronous capture.

### 15.1.4 Reader Notifications

Methods	Description
readerDidConnect()	This is generated when users plug in the fingerprint reader to the iOS device.
readerDidDisconnect()	This is generated when users disconnect the fingerprint reader from the iOS device.

### 15.1.5 Streaming Fingerprints

Methods	Description
startStreaming()	Puts the reader into streaming mode. The Reader returns the captured images using the delegate method didReceiveImage. Your code has to conform to the delegate protocol ReaderDelegate in order to receive the images.
stopStreaming()	Stops the streaming mode in order to receive the images.
setImageCaptureFormat()	Sets the image capture format. There are two types of image capture formats available. <ul style="list-style-type: none"> <li>Enhanced</li> <li>PIV</li> </ul>

## 16 Application Notes

---

### 16.1 General Fingerprint Issues

It is very important that you test your application with multiple and diverse people. The readability of fingerprints is affected by many factors including wear and tear, skin dryness, and age.

The middle finger often gives better scans than the index finger because the middle finger is typically less worn. We recommend that you enroll more than one finger for each user, preferably at least the index and middle fingers.

### 16.2 Minex certification

This current DigitalPersona Biometric SDK includes two versions of the FingerJet matching engine, versions 6 and 7.

Version 6 has a feature extractor certified by the National Institute of Standards and Technology's Minex program (<http://www.nist.gov/itl/iad/ig/minex.cfm>), as interoperable with other Minex-certified feature extractors and matchers.

Version 7 has both the feature extractor and matcher certified by the Minex program. However, the Minex-certified matcher is slower than the non-certified matcher in the version 6 FingerJet engine.

For customers who do not have a requirement to use a Minex-certified matcher, we recommend using the version 6 FingerJet engine. Specifically, for customers who use Digital Persona legacy template formats, the version 7 FingerJet engine does not provide any improvements in matching performance over version 6.

### 16.3 C/C++ Issues

The `dpfj_identify` function returns `DPFJ_SUCCESS` if it is able to do identification (i.e., the FMDs are valid and correctly formed). However that does not mean that a candidate was found. You must check the number of returned candidates to see if any actual matches were found.

The `dpfj_compare` function returns `DPFJ_SUCCESS` if it is able to do the requested comparison (i.e., the FMDs are valid and correctly formed). However that does not mean that the fingerprints matched. To check whether they matched, you must look at the dissimilarity score (0=no match, `maxint`=perfect match, `maxint-threshold` = acceptable match).





## 17 Glossary

---

### 17.1 General Terms

#### **Fingerprint**

The impression left from the friction ridges of a human finger.

#### **Fingerprint characteristics**

The biological finger surface details that can be detected and from which distinguishing and repeatable fingerprint features can be extracted.

#### **Fingerprint minutiae**

The fingerprint characteristics commonly used in fingerprint recognition systems. Fingerprint minutiae include:

**Ridge ending** – the abrupt end of a ridge

**Ridge bifurcation** – a single ridge that divides into two ridges

**Short ridge, or independent ridge** – a ridge that commences, travels a short distance and then ends

**Island** – a single small ridge inside a short ridge or ridge ending that is not connected to all other ridges

**Ridge enclosure** – a single ridge that bifurcates and reunites shortly afterward to continue as a single ridge

**Spur** – a bifurcation with a short ridge branching off a longer ridge

**Crossover or bridge** – a short ridge that runs between two parallel ridges

**Delta** – a Y-shaped ridge meeting

**Core** – a U-turn in the ridge pattern.

### 17.2 Fingerprint Data

#### **Fingerprint sample**

An analog or digital representation of a fingerprint obtained from a fingerprint capture device. See also fingerprint image.

#### **Fingerprint image**

A digital representation of a fingerprint sample encoded as a spatially mapped array of pixels. A fingerprint image is the only representation of a fingerprint sample supported by Crossmatch products, thus the terms fingerprint image and fingerprint sample are used interchangeably.

#### **Fingerprint features**

The digital representation of fingerprint characteristics.

**Fingerprint Image Data (FID)**

The binary data containing one or more fingerprint images from one or multiple fingers of the same person.

**Fingerprint Image View (FIV)**

The part of an FID that contains a fingerprint image from a single impression of a single finger.

In the majority of cases, an FID contains only one FIV.

**Fingerprint Minutiae Data (FMD)**

The digital representation of fingerprint minutiae, and optionally other fingerprint characteristics, from one or multiple fingers of the same person.

**Fingerprint Minutiae View (FMV)**

The part of an FMD that contains fingerprint features from a single impression of a single finger.

Typically FMDs contain only one FMV.

**Fingerprint template**

The fingerprint minutiae data that is stored as a result of the enrollment process.

## 17.3 Fingerprint Devices

**Fingerprint capture device**

A device that collects a signal of fingerprint characteristic and outputs a fingerprint sample. This device can consist of one or more components, including hardware and supporting software. For example, a fingerprint capture device may include a camera, photographic paper, a printer, and/or a digital scanner.

**Fingerprint reader**

A fingerprint capture device that obtains a fingerprint image by direct interaction with a finger.

## 17.4 Fingerprint Recognition Terms

**Capture**

The process of acquiring a fingerprint image from a fingerprint reader or fingerprint capture device.

**Enrollment**

The process of capturing a fingerprint image(s) for an individual, extracting fingerprint features, optionally checking for duplicates, and storing the fingerprint features (fingerprint template). See also FMD, Fingerprint Template.

**Comparison**

The function which, given two fingerprints computes a comparison score.

**Comparison score**

A comparison score is a numerical value resulting from the comparison of the fingerprint features of two fingerprints.

Comparison scores are of two types: similarity scores and dissimilarity scores.

Comparison scores are calculated using algorithms that are specific to the fingerprint recognition system and, optionally, can be normalized in order to maintain a constant score distribution of impostor verification attempts or open set identification attempts.

**Similarity Score**

See comparison score.

**Dissimilarity Score**

See comparison score.

**Fingerprint Recognition**

Verification or identification.

**Verification**

1. The process of capturing a fingerprint image from an individual,
2. Extracting fingerprint features,
3. Comparing the captured image's fingerprint features with the fingerprint template(s) of the enrollee this individual claims to be and
4. Making the match/non-match decision.

**Verification Threshold**

The maximum degree of dissimilarity that is allowed between a fingerprint image and a fingerprint template in order to make the match decision during the verification process.

**Match decision or match**

A decision that two fingerprints are from the same finger (meet the verification threshold).

**Non-match decision or non-match**

A decision that two fingerprints are not from the same finger (do not meet the verification threshold).

**Identification**

The process of

1. Capturing a fingerprint image from an individual,
2. Extracting fingerprint features and
3. Comparing the captured image's fingerprint features with the fingerprint templates from multiple individuals in order to create a candidate list of fingerprints that meet the identification threshold.

**Identification Threshold**

The maximum degree of dissimilarity that is allowed between a fingerprint image and a fingerprint template in order to call the fingerprint template a candidate in the identification process.

**Candidate**

A fingerprint template that is determined to be similar to a given FMD during identification.

## 17.5 Recognition Accuracy

The terminology below is used when discussing testing processes and reporting accuracy for fingerprint-enabled applications.

**Genuine verification attempt**

An attempt to compare fingerprint features with a fingerprint template where the fingerprint features and fingerprint template are from the same individual, and the fingerprint features and fingerprint template were captured at different times (recommended at least 2-3 weeks apart).

**Impostor verification attempt**

An attempt to compare fingerprint features with a fingerprint template, where the fingerprint features and fingerprint template are from two different people.

**False match rate (FMR)**

The ratio between the number of impostor verification attempts that produced a match decision and the total number of impostor verification attempts. See also FAR.

**False non-match rate (FNMR)**

The ratio between the number of genuine verification attempts that produced a non-match decision and the total number of genuine verification attempts. See also FRR.

**False Accept Rate (FAR)**

The application-specific measure of how often the application falsely grants a request.

In authentication applications, FAR is used in place of FMR.

**False Reject Rate (FRR)**

The application-specific measure of how often the application falsely rejects a request.

In authentication applications, FRR is used in place of FNMR.

**Verification Detection Error Trade-off (DET) curve**

A parametric curve that plots FMR on the x-axis and FNMR on the y-axis as a function of the verification threshold.

**Probe**

In simulation tests, a fingerprint used in searches (analogous to the user fingerprint used by a real application for searching).

**Gallery**

In simulation tests, a set of enrolled fingerprints (fingerprint templates) used for comparison purposes (analogous to the database of enrolled fingerprint templates used by a real application).

**Open set identification attempt**

For the purpose of testing, an attempt to compare a probe against a gallery that does not contain any impressions from any finger of the individual whose finger was used as the probe.

**Closed set identification attempt**

For the purpose of testing, an attempt to compare a probe against a gallery, where the gallery includes a fingerprint template from the same finger (from the same person) as used for the probe. The probe and the corresponding fingerprint template should be captured at different times (recommended at least 2-3 weeks apart).

**False positive identification rate (FPIR)**

Proportion of identification requests that return a candidate even though the person is not enrolled.

More precisely, for the purpose of recognition accuracy testing, the ratio between the number of open set identification attempts that returned one or more candidates and the total number of open set identification attempts.

**False negative identification rate (FNIR)**

Proportion of identification requests by enrollees that do not return the correct candidate.

More precisely, for the purpose of recognition accuracy testing, the ratio between the number of closed set *identification* attempts that did not return the correct candidate and the total number of closed set identification attempts.

**Identification detection error trade-off (DET) curve**

A parametric curve that plots FPIR on the x-axis and FNIR on the y-axis as a function of the identification threshold.

The Identification DET curve depends on the gallery size. Typically, multiple identification DET curves are shown on the same plot, one for each gallery size.



This page is intentionally left blank.

