

DigitalPersona Gold SDK

Version 3.2



digitalPersona

DigitalPersona, Inc.

© 2006 DigitalPersona, Inc. All Rights Reserved.

All intellectual property rights in the DigitalPersona software, firmware, hardware and documentation included with or described in this Guide are owned by DigitalPersona or its suppliers and are protected by United States copyright laws, other applicable copyright laws, and international treaty provisions.

DigitalPersona and its suppliers retain all rights not expressly granted.

U.are.U®, DigitalPersona® and One Touch® are registered trademarks of DigitalPersona, Inc.

Windows, Windows 2000, Windows 2003 and Windows XP are registered trademarks of Microsoft Corporation. All other trademarks are property of their respective owners.

This DigitalPersona Gold SDK Guide and the software it describes are furnished under license as set forth in the Installation Software screen "License Agreement." Except as permitted by such license, no part of this document may be reproduced, stored, transmitted and translated, in any form and by any means, without the prior written consent of DigitalPersona. The contents of this manual are furnished for informational use only and are subject to change without notice. Any mention of third-party companies and products is for demonstration purposes only and constitutes neither an endorsement nor a recommendation. DigitalPersona assumes no responsibility with regard to the performance or use of these third-party products. DigitalPersona makes every effort to ensure the accuracy of its documentation and assumes no responsibility or liability for any errors or inaccuracies that may appear in it. This document is subject to the DigitalPersona LIMITED WARRANTY and other general provisions set forth in the Appendix of this manual.

Should you have any questions concerning this document, or if you need to contact DigitalPersona for any other reason, write to:

DigitalPersona, Inc., 720 Bay Road, Suite 100, Redwood City, CA 94063 USA

Contents

Overview	5
System Requirements	6
Interface	6
High Level Functions: DpFpFns.h	6
Summary	6
Module Initialization, Termination and Settings	6
Fingerprint Reader Handling	6
Image and Feature Acquisition	6
Registration	7
Verification	7
Overview	7
Module Initialization, Termination and Settings	7
Fingerprint Reader Handling	8
Scan and Feature Acquisition	8
Registration	9
Verification	9
Device Parameters Functions: dpDevParam.h	10
Summary	10
Setting/Getting Device Parameters	10
Overview	10
Summary	11
Module Initialization, Termination, and Security	11
Database Management	11
User Management	11
Fingerprint Management	11
Authentication	11
Overview	12
User Records	12
User Authentication	13
Security	13
Feature Extraction Functions: dpFtrEx.h	14
Summary	14
Module Initialization, Termination and Settings	14
Feature Extraction	14
Overview	14
Summary	15

Module Initialization, Termination and Settings	15
Feature registration and verification	15
Overview	15
Distributing your Application	17
Windows XP Embedded Dependencies	17
Regulatory Information	18

Overview

The DigitalPersona Gold SDK can be used to develop a wide variety of custom applications related to PC or client/server access, time and attendance, and physical access. A system integrator or application developer can use the SDK to easily add the functionality of authenticating a user with fingerprint recognition to other software.

The SDK is designed to develop applications that run on the Windows platform. The standard shipping version of the SDK only works with U.are.U fingerprint readers. Custom SDKs using the core DigitalPersona Fingerprint Recognition technology can be provided to run with other fingerprint readers.

The SDK contains header files that define the API, the Fingerprint Recognition libraries, the fingerprint reader server, and Visual C++ sample code.

The DigitalPersona Gold SDK provides the application developer with two levels of APIs:

- Low-level Feature Extraction and Matching functions
- High Level authentication functions

A database module is also provided as a convenience to developers. Fingerprint templates can be stored anywhere the developer chooses, such as in an extension to a digital certificate, within a LDAP directory, in the NT SAM user database, or on a smart card.

The features are the information that describes a fingerprint. The feature extraction is the process of extracting the features (template) from the scan acquired from the fingerprint reader. The size of this template is approximately 1640 bytes. The application has the flexibility to use a smaller template size to meet stringent memory requirements for storing the template. The feature extraction process takes approximately 0.1 seconds on a Pentium® 4 CPU 3.2 GHz. The feature extraction module also returns diagnostic information about the quality of the fingerprint captured, such as image too noisy, not enough fingerprint area, or no central fingerprint region.

The matching module takes two fingerprint templates, performs a match, and verifies that the fingerprints come from the same finger. The match process takes approximately 0.01 seconds on a Pentium® 4 CPU 3.2 GHz. The application can set the security level (False Accept Rate) to be used for matches. The default security setting is 0.001% chance of false accepts. The chance of false rejects at this setting is typically 0.8%. The matching module is able to perform learning upon successful match. Learning requires extra computation and can be switched on and off by the application.

The database module can be used to store a database of user attributes and fingerprint templates. The user record contains multiple fingerprints of the user, and a protected storage area where user data such as passwords can be securely stored. The protected storage area is made available only after an authentication of the user through a correct fingerprint match. The database files are protected against unauthorized changes. Authentication is required for database updates. The “user session” functionality is provided to avoid repeated authentication.

This High Level Functions module exports high-level functions for registration and verification.

An in-built user interface is not provided. The user interface is completely under the control of the application so that consistency of application user interface can be achieved. The toolkit provides the application with status information about the on-going operation and receives from the application information about the user’s actions. This module also interacts with the U.are.U fingerprint reader server to simplify the scan acquisition and the device event monitoring. Most developers will need to use only this module.

The fingerprint reader server is a COM Server that handles multiple fingerprint readers connected to the USB ports. It dispatches notification of fingerprint capture and device events to the client applications running on the system.

DigitalPersona has given much consideration to security and user privacy issues. The fingerprint reader server sets up a challenge-response encrypted link with the fingerprint reader to securely transfer the data. Fingerprint templates are always returned encrypted from the Recognition System, and user records are stored encrypted in the database. Furthermore, registration templates cannot be matched against themselves — which means the user must provide a new sample for matching. A security key is entered by the user during installation, and is used internally as part of the

internal encryption schema of the Fingerprint Engine. Templates that are created on computers using different security keys are not compatible. Please note that the encryption functionality is not exported by the toolkit.

System Requirements

- CD-ROM drive
- Pentium-class processor (or better)
- 16Mbytes of RAM
- Universal Serial Bus (USB) Port on computer where the fingerprint reader is connected
- Windows 2000, Windows XP, Windows XP Embedded, Windows Vista or Windows Server 2003

Interface

High Level Functions: DpFpFns.h

Language: ANSI C
Notes: prefix FT_

Summary

Module Initialization, Termination and Settings

FT_getVersionInfo: gets the software version of the dpFpFns module.
FT_init: initializes the dpFpFns module.
FT_setupPlugAndPlayMonitor: sets callback to monitor plug and play.
FT_setupUllink: sets up the link with the UI to provide feedback and get user actions.
FT_getSettings: gets the current dpFpFns settings.
FT_terminate: terminates the dpFpFns module.
FT_createContext: creates context.
FT_closeContext: destroys the resource allocated for the given context.
FT_setSecurityLevel: sets the security level of the Fingerprint Recognition System.
FT_getSecurityLevel: gets the security level of the Fingerprint Recognition System.

Fingerprint Reader Handling

FT_getNumDevices: returns the number of devices connected to the computer.
FT_getDeviceList: gets the device list.
FT_getConnectedDeviceInfo: gets the connected device info.
FT_getDeviceInfo: gets the connected device extended device info.
FT_connectDevice: initializes the fingerprint reader.
FT_disconnectDevice: terminates the communication with the fingerprint reader.
FT_startMonitoringDevice: specifies event to be signaled when a scan is available.
FT_stopMonitoringDevice: stops monitoring device.
FT_flushFpImages: flushes any scans for a specified device.

Image and Feature Acquisition

FT_getFeaturesLen: returns minimum and recommended lengths for a given features' type.
FT_acquireImage: acquires a scan from the specified fingerprint reader.
FT_getImagePixels: gets image pixels from the scan acquired from the fingerprint reader.
FT_acquireFeatures: extracts features from the scan on the fingerprint reader.

Registration

FT_register: performs the registration operation.

Verification

FT_verify: decides whether print on fingerprint reader matches given registration print.

FT_verifyEx: decides whether print on fingerprint reader matches given registration print.

Overview

Module Initialization, Termination and Settings

FT_getVersionInfo returns the software version of the dpFpFns module.

FT_init initializes the toolkit and the fingerprint reader server, and must be called before any other function is called.

FT_createContext creates a context to be used with high level functions.

FT_setupUIlink is used to set up the link between a device and the User Interface (UI), via an argument of type FT_UI_LINK_PT that contains callback functions and fingerprint reader parameters. The toolkit does NOT handle the UI, which is completely under the control of the application. The toolkit provides the application with status information about the on-going operation and receives from the application information about the user's actions. The communication between the toolkit and the application is accomplished through "callback functions", that is, functions provided by the application and called by the toolkit when needed. In some cases, the address of a callback function could be NULL.

The callback functions obtained by the toolkit via FT_setupUIlink are of the following three types:

- **FT_GETACTION**
The toolkit calls this function with an argument that provides the application with status information (for instance, "waiting for a scan"). The application, based on the status and on the dialog with the user, returns an action to the toolkit (for instance, FT_CANCEL). The toolkit calls this function frequently, so that the application can be responsive to Windows. Note that a FT_GETACTION function MUST be specified (its address cannot be NULL).
- **FT_GET_DISPLAY_BUF**
During the registration and verification operations, the toolkit acquires one or more scans. For each of them, it uses this callback function to ask the application for a display buffer. When a scan has been acquired and the toolkit is ready to fill the display buffer, it notifies the application by calling the FT_GETACTION function, with the status code = FT_READY_TO_FILL_BUF. Then, the toolkit copies the image into the buffer and notifies the application by calling the FT_GETACTION function, with the status code = FT_BUF_FILLED. The application can in turn display the image to provide the user with feedback. For increased flexibility, the application can specify the size of its display buffers at initialization time. (See FT_DISPLAY_BUF_SIZE.) The toolkit takes care of any re-sampling of the image if needed. If the application intends to do registration, it must always provide the FT_GET_DISPLAY_BUF callback function since user feedback is critical. For verification only, this callback function is optional (its address may be NULL).
- **FT_RELEASE_DISPLAY_BUF**
The toolkit calls this function when it has finished acquiring a scan to release a specific display buffer. This callback function must be specified (address not NULL) whenever the FT_GET_DISPLAY_BUF is specified.

FT_getSettings returns the current dpFpFns module settings.

The security level can be modified and obtained by using FT_setSecurityLevel and FT_getSecurityLevel respectively.

FT_closeContext closes the context.

FT_terminate terminates the dpFpFns module and releases all resources.

Fingerprint Reader Handling

The Fingerprint Toolkit can handle multiple devices connected to the computer simultaneously. The application may obtain the list of available devices as well as be notified if devices are plugged or unplugged from the computer.

FT_getNumDevices returns the number of devices plugged in the computer.

FT_getDeviceList gets the list of the devices plugged in the computer.

FT_getConnectedDeviceInfo gets information about the capabilities of a given connected device.

FT_setupPlugAndPlayMonitor is used to specify the plug and play monitor callback function of type FT_PLUG_AND_PLAY_MONITOR. The application is notified if a fingerprint reader has been plugged or unplugged.

FT_connectDevice initializes the fingerprint reader specified by the given ID.

FT_disconnectDevice releases all the resources allocated for communication with the fingerprint reader.

FT_startMonitoringDevice allows the caller to specify an event to be signaled when there is a finger activity on the fingerprint reader (finger arriving, finger leaving, scan received). FT_stopMonitoringDevice stops the monitoring.

FT_flushFpImages flushes any scans in queue for a given device.

Scan and Feature Acquisition

A raw fingerprint image can be acquired by using FT_acquireImage as output by the fingerprint reader. This image can be used later by features extraction or for obtaining image pixels.

Image pixels can be obtained by using FT_getImagePixels. The output is a grey scale one byte per pixel image. To calculate the size of the fImagePixels buffer for a particular fingerprint reader, one should obtain the FT_DEVICE_INFO structure and get the product of the width and height of the imgSize field.

The features can be of three types: pre-registration, registration or verification. The registration features contain the template stored for future matching and are built based on pre-registration features. The verification features are matched against registration features.

The toolkit returns the optimal and minimum sizes needed to hold the features. The application should use the optimal size unless specific memory requirements require otherwise. Keep in mind that by using larger features you can increase the performance of the verification at the expense of storage. It is advisable to use quite large features for verification since they, unlike the registration features, do not need to be stored.

The fingerprint toolkit is capable of learning. Larger features are needed to enable learning. Learning can potentially improve the recognition performance, at the expense of increased processing time and storage. If learning is enabled, the features may change over time. Their sizes remain constant.

FT_acquireFeatures is used to acquire fingerprint reader features of a given type (pre-registration, registration or verification) with a specified length. In addition to the features, it returns measures of image quality and feature quality. FT_acquireFeatures accommodates different storage requirements and different levels of security via the input parameter featuresLen. Use FT_getFeaturesLen to get the recommended and minimum features length.

Registration

Registration is done with `FT_register`.

The main steps of the registration operation are:

1. Acquire a fingerprint scan of adequate quality.
2. Process the image to obtain the features that uniquely describe the fingerprint. Such features are called “pre-registration features”. They are different from features produced in step 3, and cannot be used for verification. The pre-registration features are rejected if they are not of adequate quality. Step 1 and step 2 are repeated until a given number of adequate pre-registration features are obtained (the operation can also be cancelled). This number is provided in the toolkit settings. (See `FT_SETTINGS` returned by `FT_init` and `FT_getSettings`.)
3. Produce the “registration features” from the set of pre-registration features, after verifying that the prints match sufficiently well. The registration features can then be stored for later use in verification.

The toolkit does NOT save fingerprint bitmaps.

`FT_register` does not handle the user interface. It uses the callback functions and parameters specified with `FT_setupUllink` to provide feedback to the user.

Verification

`FT_verify` and `FT_verifyEx` acquire a fingerprint from the fingerprint reader and decide whether it matches the given registration features. The level of security is set with the function `FT_setSecurity`.

The main steps of the verification operation are:

1. Acquire a fingerprint scan of adequate quality.
2. Process the image and extract its features (called the “verification features”) that uniquely describe the fingerprint.
3. Determine whether the verification features match the registration features.
4. If `doLearning` is true, the registration features might be updated.

`FT_verify` does not handle the user interface. It uses the callback functions specified with `FT_setupUllink` to allow the application to provide feedback to the user.

Device Parameters Functions: dpDevParam.h

Language: ANSI C
Notes: prefix FT_

Summary

Setting/Getting Device Parameters

FT_SetParameter: sets a device parameter.
FT_GetParameter: gets a device parameter.

Overview

These two functions FT_SetParameter and FT_GetParameter are available to "personalize" the fingerprint reader. They can be used to write private data to the reader, which is stored as a hash, and to read the stored hash. dpFTDeviceKey.exe is included in the list of sample code applications. It demonstrates utilization of these two functions.

Database Functions: DpDbase.h

Language: ANSI C
Notes: prefix DB_

Summary

Module Initialization, Termination, and Security

DB_getVersionInfo returns the software version of the database module.
DB_init: initializes the database module.
DB_setSessionIdDuration: sets the duration for session ID of the given context.
DB_getSessionIdDuration: gets the duration for session ID of the given context.
DB_setSecurityLevel: sets the security level of the given context.
DB_getSecurityLevel: gets the security level of the given context.
DB_terminate: terminates the database module.

Database Management

DB_createDBA: creates a database, using ASCII for database name and path arguments.
DB_createDBW: creates a database, using UNICODE for database name and path arguments.
DB_deleteDBA: deletes a database, using ASCII for database name and path arguments.
DB_deleteDBW: deletes a database, using UNICODE for database name and path arguments.
DB_openDBA: opens an existing database, using ASCII for database name and path.
DB_openDBW: opens an existing database, using UNICODE for database name and path.
DB_changeDBOpenMode: changes between read/write and read-only modes.
DB_saveDB: saves the open database and leaves it open.
DB_closeDB: saves and close the database.
DB_compactDB: eliminates deleted records from the database.
DB_readDBHdr: reads the database header, and any application defined database info.
DB_writeDBInfo: writes application defined database info.

User Management

DB_newUser: adds a user to the database.
DB_changeUserType: changes user type to DB_ADMINISTRATOR or DB_REGULAR.
DB_changePasswd: changes user's password.
DB_delUser: deletes a user.
DB_getUserId: gets the user ID corresponding to a user name.
DB_topUser: goes to the first user in the database.
DB_nextUser: gets the user ID, name, and user info of the next user in the database.
DB_readUserData: gets user's name and info with the given user ID.
DB_writeUserInfo: writes application defined user info in a user record.
DB_readPrivateStorage: gets the user's private storage.
DB_writePrivateStorage: writes the user's private storage.

Fingerprint Management

DB_addFinger: adds registration features to a user record, linked to a given finger.
DB_readFinger: reads registration features from a user record.
DB_setDfltFinger: sets the default finger used for verification.
DB_delFinger: deletes a registered finger from the database.

Authentication

DB_authenticate: determines if given features or password match those of a given user.

DB_authenticateEx: determines if given features or password match those of a given user.
DB_extendSessionIdDuration: resets the time for the given session ID.
DB_releaseUserSessionId: terminates the given session ID.

Overview

The fingerprint database contains individual user identification information, and associates a set of registered fingerprint(s) with each user. The database has a header containing general information about the database, such as the number of users or time of last update. In addition, application-specific database information, or “database info”, can be written with DB_writeDBInfo; its length is specified at the time of database creation. Both the header and the database info can be read with DB_readDBHdr. A database format version is included in the header, in order to make it straightforward in the future to read an old database even if the toolkit that created it has been replaced by a newer version.

Creation of a fingerprint database is performed with DB_createDBA or DB_createDBW, depending on whether the provided database name and path are in ASCII or UNICODE. These functions allow for the specification of the sizes of various record types, depending on the needs of the application. An already existing database can be opened with DB_openDBA or DB_openDBW, and deleted with DB_deleteDBA or DB_deleteDBW, where the choice between the ASCII and UNICODE versions should be the same as when it was created. All other character strings, such as user names, are in UNICODE.

Although several users are allowed simultaneous access to the same database in read-only mode, only one read-write access to a given database is permitted at a given time. Alternating between read-only and read-write open modes can be conducted within a given session, with DB_changeDBOpenMode, provided that no more than one open database has read-write status at a given time.

The fingerprint database is stored in encrypted form in two files: the user file, with extension .fpu, and the registration fingerprint file, with extension .fpr. The fingerprint scans are not saved on disk. Access to a database record is restricted to those authorized to access that particular record by requiring session IDs. (See Security section.)

Other functions, which apply to the whole database, are DB_saveDB, DB_closeDB, and DB_compactDB, the last of which is described in the USER RECORDS section.

User Records

The user record links a user to their fingerprint features as well as to their keys. Initial acquisition of the fingerprint scans is accomplished with the dpFpFns library. Subsequent feature extraction and processing is then performed with the dpFpFns library or dpFtrEx library. The resulting processed fingerprint features are then added to the user record. A user record also contains a password, stored in hashed form. The user record may also contain other information, depending on the application, as explained in the next paragraph. The entire record is stored in the database in encrypted form. Each user record has type DB_ADMINISTRATOR or DB_REGULAR, which determines the access control to the data.

The user record may contain “user info”, which is used for application-dependent identification details; its length is specified at the time the database is created. It is written with DB_writeUserInfo and read with DB_readUserData. Private storage can also be associated with the user, and its length is similarly specified at the time of database creation. Private storage can be written and read with DB_writePrivateStorage and DB_readPrivateStorage. Private storage can only be retrieved by the user it belongs to or by an administrator, and only after the person successfully authenticates himself. (See User Authentication section.)

New user records are created with DB_newUser. The stored user records are alphabetically sorted. A user record is accessed through its USERID. The USERID corresponding to a user name can be obtained with DB_getUserId. Scanning the database is accomplished by starting at the first database record with DB_topUser, and then sequentially accesses user records with DB_nextUser.

After a user is added, any or all of their fingers can be registered. Once the registration features associated to a fingerprint have been obtained through the dpFpFns, dpFtrEx and dpMatch libraries, they can be added to a user record

with `DB_addFinger`. `DB_setDfltFinger` is used to change the default finger used for verification; this is the finger that will be used if `DB_DEFAULT` finger is specified when authenticating.

Registration features for a particular finger can be deleted with `DB_delFinger`. A user can be deleted from the database along with their corresponding fingerprint information with `DB_delUser`. Deleted records will not be physically removed from the database, but they will no longer be accessible by the `DB_nextUser` function. The space occupied by deleted records will not be reused. Deleted records are physically removed from the database with the function `DB_compact`.

The password can be changed with `DB_changePasswd`, but only after the user it belongs to, or an administrator, is successfully authenticated.

Only an administrator successfully authenticated can change the user type by using `DB_changeUserType`.

User Authentication

A user can be authenticated with `DB_authenticate`. The given verification features, obtained from the `dpFpFns` and `dpFtrEx` libraries, are compared to the registered ones. If the verification is successful, `DB_authenticate` returns the user and finger keys as well as the private user storage. If a finger is specified, then the verification features will be matched only against the registration features of that finger. If the finger argument is `DB_DEFAULT`, then the verification is performed against the features of the default finger. If the finger argument is `DB_UNKNOWN`, then the verification will be performed against all the features registered for that user, and if a match occurs, the finger key structure will show which finger matched.

`DB_authenticate` can also verify a password in addition to verification features. In this case, the password is hashed and compared to the hashed password stored in the user record. If they match, the verification is successful.

`DB_authenticate` requires the `USERID` of the user being authenticated. Therefore, it cannot be used directly to identify an unknown user. However, unknown user identification can be performed, by calling `DB_authenticate` repeatedly, until either a successful verification occurs or all valid `USERIDs` fail.

Security

In order to prevent inappropriate user access or tampering, all records in the user file (*.fpu) and all features in the registration fingerprint file (*.fpr) are encrypted. The encryption key is derived from an embedded key and the installation key and is therefore installation-dependent. The toolkit uses cryptographic functions internally, but they are not exported, and therefore not available to the application.

The toolkit uses a “session ID” to restrict database operations to authorized users. A session ID is returned by `DB_authenticate` if authentication is successful, and it can then be used for a limited period of time to “prove” to other functions that the person currently using the application is authorized to do the attempted action. It is encrypted, so that a user cannot modify it to pretend to be someone else. A timestamp is also added before encryption, so that session ID’s cannot be saved for later use. Extension or termination of a current valid session ID is accomplished with `DB_extendSessionIdDuration` or `DB_releaseUserSessionId`, respectively.

All functions that modify sensitive information in the database require a session ID, as does `DB_readPrivateStorage`. If the session ID is for a user of type `DB_REGULAR`, no changes to any other user record are allowed, and some restrictions apply even to changes of the user’s own record: for example, they cannot change their type to `DB_ADMINISTRATOR`. Users of type `DB_ADMINISTRATOR` are authorized to use all functions in the library.

User records contain backup passwords, chosen by the user at registration, in order to allow for user authentication if fingerprint authentication cannot be used. Passwords are hashed and only the hash is stored in the user record.

Feature Extraction Functions: dpFtrEx.h

Language: ANSI C
Notes: prefix FX_

Summary

Module Initialization, Termination and Settings

FX_getVersionInfo returns the software version of the feature extraction module.
FX_init: initializes the feature extraction module.
FX_terminate: terminates the feature extraction module.
FX_createContext: creates a feature extraction context.
FX_closeContext: closes a feature extraction context.

Feature Extraction

FX_getFeaturesLen: returns the minimum and recommended length of the features.
FX_extractFeatures: extracts the features from the given scan.
FX_getDisplayImage: returns an image prepared for display.

Overview

The dpFtrEx module contains code to perform feature extraction. This is the process of deriving from the fingerprint information expressed in a form that is suitable for comparing prints and determining if they come from the same finger.

FX_init initializes the module, and must be called before any of the other functions are called. FX_getVersionInfo returns the software version and FX_terminate terminates the dpFtrEx module and releases all resources.

The feature extraction modules maintain one or more contexts for each caller. A context can be created by calling FX_createContext, and eventually released with FX_closeContext. A handle to the context has to be passed to FX_extractFeatures and FX_getDisplayImage.

FX_extractFeatures is the function that extracts the features from the image, which is passed as one of the arguments.

FX_getDisplayImage is the function that prepares the image for display.

Matching Functions: dpMatch.h

Language: ANSI C
Notes: prefix MC_

Summary

Module Initialization, Termination and Settings

MC_getVersionInfo returns the software version of the feature extraction module.
MC_init: initializes the matching module.
MC_terminate: terminates the matching module.
MC_createContext: creates a matching context.
MC_closeContext: closes a matching context.
MC_getSettings: gets the matching module settings.
MC_setSecurityLevel: sets the security level of the matching module.
MC_getSecurityLevel: gets the security level of the matching module.

Feature registration and verification

MC_getFeaturesLen: returns the minimum and recommended length of the features.
MC_generateRegFeatures: generates registration features.
MC_verifyFeaturesEx: performs verification of registration and verification features.

Overview

The dpMatch module contains code that compares the registration template (obtained by processing pre-registration features) and verification features (extracted from the fingerprint scan) and calculates a score that indicates how likely it is that they come from the same finger. Most of the functions must be called in a particular context, which is specified by passing a context handle as the first argument.

MC_init initializes the module, and must be called before any other functions are called.

MC_terminate closes the dpMatch module and releases all resources.

MC_getVersionInfo returns the software version.

MC_createContext creates a context, and returns a handle to it.

MC_closeContext is used to close a context.

The dpMatch module has settings, which are returned by MC_getSettings in a structure of type MC_SETTINGS. It provides the number of pre-registration features required for registration. The settings are read only.

A registration template is created by MC_generateRegFeatures. It requires the pre-registration features extracted from multiple samples of a single finger as input. It constructs and returns the registration template that will be used to verify the fingerprint in the future.

The XTF (eXtended Template Format) registration templates contain additional information needed for a more accurate verification. The template size is bigger than that of the basic template (about 1640 instead of 330 bytes). The matching time may slightly increase. If you do not have storage limitations, it is strongly recommended to use XTF registration.

During verification, the recognition engine may “learn” additional information about the fingerprint, and may update the registration template, so that future verifications will be performed with higher accuracy. In order to use this feature, learning must be enabled during registration.

Different options for registration may be set with `mcRegOptions` by combining bit masks with the bitwise OR operator. Currently there are four options (bit masks) available:

1. `FT_DEFAULT_REG`: enables the default combination of options which ensures the optimal performance. This default set of options may be changed in future releases. Currently XTF registration is set by default.
2. `FT_DISABLE_DEFAULT_REG`: disables the implicit use of the default registration options. It gives the caller complete control over the registration options used. It can be used in conjunction with the other bit masks. In this case, the options explicitly set in the resulting mask will be used for registration. For example, using the `FT_DISABLE_DEFAULT_REG | FT_ALLOW_LEARNING` bit mask will guarantee that only learning will be set - regardless of the default options implicitly set by current engine.
3. `FT_ENABLE_XTF_REG`: enables the XFT registration.
4. `FT_ALLOW_LEARNING`: enables learning during fingerprint verification.

`MC_verifyFeaturesEx` compares a registration template to verification features and returns TRUE if the matching score is high enough (given the current security level). If learning is enabled, it attempts to “learn” and may update the registration template

Both registration and verification are performed with a given accuracy. `MC_getSecurityLevel` returns the current security level. `MC_setSecurityLevel` can be used to modify the security level, according to the accuracy required by the application. The security constant `HIGH_SEC_FA_RATE` refers to 0.0001% (1/1,000,000) FAR, `MED_SEC_FA_RATE` refers to 0.0010% (1/100,000) FAR, and `LOW_SEC_FA_RATE` refers to 0.0100% (1/10,000) FAR. `MED_SEC_FA_RATE` is set by default.

Distributing your Application

After you have developed your application and want to distribute it, or test it on a clean computer, you will need to purchase the DigitalPersona Gold Integrator Package. This package includes the U.are.U Fingerprint Reader and the DigitalPersona Gold Fingerprint Recognition Software CD, containing the installer of the runtime.

To install the DigitalPersona Gold Fingerprint Recognition Software silently follow the instruction below.

To use the default encryption key:

1. Copy all the files from the \Install folder located on the product CD to the hard drive.
2. Open the Setup.ini file. On the last line you will see the following:
;cmdline=/qn FT_1_KEY=54df8c46a66ba14fdd8f0152cce9e65f FT_2_KEY=d6c49aefd2ff0c3e54bf942a588faa7b /I
Remove the “;” at the beginning of the last line to uncomment the silent install command.
3. Save the modified Setup.ini file.
4. Run Setup.exe, which will run in silent mode.

To use a custom encryption key:

1. On a clean computer, run Setup.exe located on the DigitalPersona Gold Fingerprint Recognition Software CD.
2. Towards the end of the installation, setup will prompt you to type a 16 character encryption key. Type your custom key (any character from a to Z, and from 0 to 9).
3. Finish the installation.
4. Go to the Registry Editor and locate the following key:
Hkey_LocalMachine\Software\DigitalPersona\fp_toolkit
5. Open the key and copy the Value data of FT_1 and FT_2 and paste them in the last line of the Setup.ini file.
6. Save the modified Setup.ini file.
7. Run Setup.exe, which will run in silent mode.

Windows XP Embedded Dependencies

DigitalPersona Gold SDK and Fingerprint Recognition Software 3.0 are certified to run on Windows XP Embedded. For the list of Windows XP Embedded dependencies please see the Gold 3.0 Windows XPE Dependencies.xls file located in the Docs folder on the product CD.

Regulatory Information

Any changes or modifications not expressly approved by DigitalPersona could void your authority to operate this equipment.

The U.are.U Fingerprint Reader has been tested and found to comply with the limits for a Class B digital device under Part 15 of the Federal Communications Commission (FCC) rules, and it is subject to the following conditions:

a) It may not cause harmful interference, and b) It must accept any interference received, including interference that may cause undesired operation.

This device conforms to emission product standards EN55022(B) and EN50082-1 of the European Economic Community and AS/NZS 3548 Class B of Australia and New Zealand.

This digital apparatus does not exceed the Class B limits for radio noise emission from digital apparatus as set out in the radio interference regulations of the Canadian Department of Communications

Le présent appareil numérique n'émet pas de bruits radioélectriques dépassant les limites applicables aux appareils numériques de Classe B prescrites dans le règlement sur le brouillage radioélectrique édicté par le Ministère des Communications du Canada.