NIST ELFT-EFS
Evaluation of Latent Fingerprint Technology — Extended Feature Sets
Evaluation 2
Test Plan / API / Schedule

12 April 2010

Contents

1 Overview

The NIST Evaluation of Latent Fingerprint Technology — Extended Feature Sets (ELFT-EFS) is an independently administered technology evaluation of latent fingerprint feature-based matching systems. ELFT-EFS is being conducted by the National Institute of Standards & Technology (NIST).

ELFT-EFS is a complement to NIST's Evaluation of Latent Fingerprint Technology (ELFT) testing program. The ELFT evaluations to date have focused solely on automated feature extraction and matching (AFEM) in the context of latent fingerprint identification.

ELFT-EFS will evaluate the accuracy of latent matching using features marked by experienced human latent fingerprint examiners. The purpose of this test is to evaluate the current state of the art in latent feature-based matching, by comparing the accuracy of searches using images alone with searches using different feature sets. The features sets will include the current IAFIS latent feature set, and different subsets of the Extended Feature Set (EFS) features proposed by CDEFFS[1]. A key result of the test is to determine when human feature markup is effective. Because human markup is expensive in terms of time, effort, and expertise, there is a need to know when image-only searching is adequate, and when the additional effort of marking minutiae and extended features is appropriate.

The following summarizes the planned test:

- The evaluation will involve 1:N searches using latent 1000ppi images provided with human markup of EFS features.
- Exemplars for the gallery will be images only. Exemplars will be 500ppi.
- The test will be an SDK-type test, in that participants will provide software, and all processing will take place on NIST hardware.
- Different tests will be run for the following search types:
  - Image only
  - Image with region of interest markup
  - Image with minutiae (IAFIS EFTS LFFS equivalent)
  - Image with EFS features
  - Minutiae only (IAFIS EFTS LFFS equivalent)

Test results will be made publicly available in a NIST report after the conclusion of the test.

2 Participation

Participation in Evaluation 2 is open to all developers of latent fingerprint identification systems.

All systems must comply with the API outlined in Section 5.4. Anonymous participation will not be permitted. The Application form[2] includes details regarding application and qualification.

3 Data

3.1 Datasets

Validation Dataset

A Validation Dataset will be provided to participants before the evaluation to verify the correct operation of participants' software before and after delivery to NIST.

Evaluation Dataset

The latent and exemplar images and features in Evaluation #2 will be similar but not identical to those in Evaluation #1. The Evaluation Dataset will contain sequestered data, formatted in the same manner as the Validation Dataset. The Evaluation Dataset will contain Privacy Act or FOIA Protected Information and will not be released to the participants or the public. The Evaluation Dataset will to the extent permitted by law be protected under the Freedom of Information Act (5 U.S.C 552) and the Privacy Act (5 U.S.C. 552a) as applicable.

[1] *CDEFFS is the ANSI/NIST Committee to Define an Extended Fingerprint Feature Set. The current working draft of the Extended Fingerprint and Palmprint Features document can be found at http://fingerprint.nist.gov/standard/cdeffs/.*

[2] *The Application form can be found at http://fingerprint.nist.gov/latent/elft-efs/*

3.2 Format

All images and data will be contained in ANSI/NIST files. All images will be 8-bit grayscale.

Each latent ANSI/NIST file in the evaluation will contain one Type-1 record, one Type-2 record, zero or one Type-9 records, and one Type-13 record. All latent images will be in Type-13 records, in uncompressed format.

Each exemplar ANSI/NIST file in the evaluation will contain one Type-1 record, and ten Type-14 records (one for each finger, with finger positions identified). All exemplar images will be in Type-14 records. 500ppi exemplar images will be compressed using WSQ.

3.3 Features

Files containing exemplars will not have any features defined: no Type-9 record will be present.

Files containing latents may or may not have any features defined: zero or one Type-9 records will be present. There will be tests comparing the accuracy of two primary types of searches:

- Image-only searches, in which the latent image will not be accompanied by a type-9 record.
- Feature-based searches, in which the latent image will be accompanied by a type-9 record with features defined in fields 9.300-9.372, formatted in accordance with "Data Format for the Interchange of Extended Fingerprint and Palmprint Features," abbreviated here as the "EFS Spec" (Extended Feature Set Specification). The test will evaluate different combinations of EFS fields, so not all EFS fields may be present in any given search. The subsets of features used (defined as Subsets LA-LG) are defined in Section 7.

Note: The current EFS Spec version is 0.4 (June 2009).

All of the latent IAFIS/EFS features will be provided with feature markup by human experts. Note that all human markup will be conducted outside of ELFT-EFS and is not part of the evaluation.

Note also that conformance testing of automatic extraction of CDEFFS features is not part of this test. In other words, the evaluation will not be measuring how close automatically extracted features are to examiner created features. Automated algorithms can use the extended features defined for a latent search without explicitly computing them for the exemplar image, and thus it must be emphasized that automated extraction of the extended features on the exemplar is not necessarily the only nor the best way to use this information. For example, an examiner may mark an area as a scar; for the exemplar, the matcher would not necessarily have to mark the area as a scar, but may use that information to match against a corresponding area with many false minutiae and poor ridge flow.

3.4 Resolution

All latent images will be 1000 pixels per inch.

Exemplar images will be at 500 pixels per inch. This resolution will be contained in field 14.009 (Horizontal pixel scale), which will be identical to field 14.010 (Vertical pixel scale).

3.5 Dimensions and orientation

Latent fingerprint images may vary from 0.3"x 0.3" to 2.0" x 2.0" (width x height), all at 1000ppi. 1st & 3rd quartiles are about 700-1200 pixels (width) or 900-1400 pixels (height).

Exemplar images will be approximately upright (in the same orientation as they were captured).

Neither latent nor exemplar images will be larger than 2.0" in either width or height.

When orientation is known in advance or discernable by an examiner, latent fingerprint images will not vary in orientation from upright more than ±45°. However, there are images in the Evaluation Dataset for which orientation cannot be determined in advance, and all possible orientations (±180°) must be considered equally likely. Images from latent subtests LB-LG will

include the orientation direction and uncertainty fields (9.301). Images from latent subtest LA will not.

3.6 Exemplar types

All exemplars will include rolled or plain (segmented slap) fingerprints. The impression types will include optical livescan and inked paper sources. The impression type will be noted in field 14.003.

Exemplars will always include all ten fingers, and are therefore referred to here as a 10-finger exemplar set (also commonly called a ten print set).

Note that a 10-finger exemplar set will consist of either ten rolled prints, or ten plain prints.

In some cases, multiple sets of 10-finger exemplar sets associated with one person will be included in the gallery. This association will be made explicit in the exemplar enrollment stage: at the time of enrollment, exemplars that are known to belong to the same person will always share the same subject ID.

3.7 Finger positions

Exemplars will be provided in complete 10-finger sets, all contained within a single ANSI/NIST file, with finger positions noted.

The finger positions for latents will not be noted – no searches will be restricted to specific fingers.

3.8 Dataset size

The largest size gallery used for Evaluation 2 will contain 100,000 subjects having two 10-finger exemplar sets (rolled and plain impressions) per subject.

The total number of unique latent images is approximately 1,100, with the number of latent searches based on section 4.2.

4 Evaluation Criteria

4.1 Performance Metrics

Performance metrics will be based on rank and matcher score:

- Rank will be reported by the number of true matches reported in each position in the candidate list. For example, the Rank-1 metric is the proportion of searches in which the correct mate appears in the top position on the candidate list. CMC[3] curves will also be reported to show how many latent images are correctly identified at rank 1, rank 2, etc. A CMC is a plot of identification rate vs. recognition rank. Identification rate at rank k is the proportion of the latent images correctly identified at rank K or lower. A latent image has rank k if its mate is the kth largest comparison score on the candidate list. Recognition rank ranges from 1 to 100, as 100 is the (maximum) candidate list size specified in the API.

- Matcher score metrics are evaluated in terms of DET/ROC[4] performance, by plotting False Positive Identification Rate (FPIR) and False Negative Identification Rate (FNIR) for all score values. Note that this approach requires that a given matcher score be comparable between different latent searches. Both the absolute matcher score and the probability of true match values (see Section 5.6) will be used for DET analysis.

[3] *Cumulative Match Characteristic*
[4] *Detection Error Trade-off/Receiver Operating Characteristic*

4.2 Evaluation Subtests

The Evaluation is composed of the following subtests. For precise definitions of which features will be present for each subtest: see Section 7. All latents in each subtest may or may not be searched against all exemplars (galleries). Participants have the option of specifying which specific subtest combinations they wish to be tested on.

- Latent Subtests
  - LA – image only
  - LB – image + ROI
  - LC – image + ROI + Pattern Class + Quality Map
  - LD – image + IAFIS/EFTS equivalent features
  - LE – image + baseline EFS
  - LF – image + baseline EFS + Skeleton
  - LG – IAFIS/EFTS equivalent features only
- Exemplar Subtests
  - E1 – 100,000 subjects; 1 set of 10 rolled and 1 set of 10 plain impressions each; 500ppi

4.3 Reporting of Results

The ELFT-EFS Final Report will contain descriptive information concerning the evaluation, descriptions of each experiment, aggregate test results across all participants, and individual test results for each participant. All results will be reported for each participating system, with the exception of results for different combinations of EFS features. Because not all participating systems may implement all of the EFS features, results from those evaluations will be stated in generic terms so that participants cannot deduce which features are used by other systems.

Note that the application form stipulates that each participant consents to the disclosure of its performance.

Enrollment, feature extraction and search timing information will also be reported, with the explicit caveat that speed of execution, for both enrollment and latent search, is of secondary importance. The report will specify the hardware specifications used in the evaluation, and will also note that operational latent searching algorithms are likely to be implemented in more sophisticated hardware.

5 Latent Matching Software

5.1 Overview

Participants shall submit a set of SDKs (Software Development Kits) that provide the interfaces defined by the ELFT-EFS API specified below. The SDKs shall be provided as static or dynamic libraries to run on the NIST platform specified below. The ELFT-EFS API (Application Programmer Interface) is modeled after the API from ELFT Phase 2. The most notable differences from the ELFT Phase 2 API are that the exemplar and latent images and data provided to the SDK will be contained in ANSI/NIST files, and exemplar feature extraction will process a single exemplar per invocation (instead of the complete gallery). Also, the ELFT-EFS API specifies operational time limits on a per-processor core basis, rather than per-machine.

Each participant shall submit

- one SDK for exemplar feature extraction and exemplar enrollment
- one SDK for latent feature extraction
- one SDK for latent 1-to-N search

NIST recognizes the proprietary nature of the participant's software and will take all reasonable steps to protect this. The software submitted will be in an executable library format, and no algorithmic details need be supplied. NIST agrees not to use the Participant's software for purposes other than indicated above, without express permission by the Participant.

5.2 Test Platform

The NIST ELFT-EFS Evaluation test platform consists of an array of blade servers having a hardware configuration similar to:

Processor

- Dual 2.8 GHz/1MB Cache, Xeon (dual-core)
- 800 MHz Front Side Bus for PE 1855

Memory

- 16GB RAM (15GB available to applications)

Secondary storage

- 300GB 15K RPM Ultra SCSI Hard drives

The operating systems available (in order of preference) are:

- RedHat Enterprise Linux Server 5.1 (64-bit)
- Windows 2008 Server (64-bit)
- (Windows Server 32-bit may be available on request)

The available RAM for 64-bit SDKs will be no more than 15GB total. The available RAM for 32-bit SDKs will be no more than 3GB per process.

5.3 Execution protocol

Each SDK tested will be allocated multiple blades/cores from the array, along with a subset of the test data in order to maximize (time) efficiency through parallel operation.

Each SDK instance assigned to an individual blade or core will operate on a subset of the data, using individual data copies (as needed) from a local storage device.

For purposes of execution, there are two classes of SDKs, (1) sequential and (2) multithreaded. And each class the SDK may utilize either 32 or 64-bit execution mode. *Note that each SDK submitted (i.e. either of the two SDKs per participant) may be of a different class and execution mode. For example, the Exemplar feature extraction / enrollment SDK may be sequential 32-bit and the Latent feature extraction / search SDK may be multithreaded 64-bit.*

It is highly recommended that SDKs implement multithreading using 64-bit execution mode. However, if some participants are unable to submit multithreaded or 64-bit SDKs, we support other modes of operation as outlined below.

5.3.1 Sequential

An advantage of sequential (i.e. non-multithreaded) SDKs is the ability to "manually" parallelize SDK execution for a given test by executing multiple instances per blade server (e.g. one per core). A potential drawback is that individual 64-bit SDK instances have the potential to over-allocate available RAM, which may result in "swapping," decreasing overall execution speed. Another potential drawback is contention for resources given that each instance is executing independently (i.e. without coordinated resource usage). For this reason NIST does not recommend the submission of sequential SDKs.

As a simple example, the execution of a sequential SDK for a subtest requiring M latent searches against N exemplars (i.e. Gallery size N), may allocate M searches amongst K available cores such that each core is executing M/K searches total. The primary choice here is whether or not to allocate all cores available on a given blade server, or a subset thereof. How much memory is allocated by the SDK (limited by whether it is 32 or 64-bit mode) is a primary consideration.

Sequential SDKs which run in 32-bit execution mode shall have access to no more than 3GB per process. NIST will execute four (4) SDK instances (one instance per core) on each available blade server, in order to maximize processor and memory utilization.

Sequential SDKs which run in 64-bit execution mode shall have access of up to 15GB per process, and the participant should inform NIST at submission time as to the SDK's memory usage requirements. It is strongly recommended that the SDK perform most efficiently when executed as four (4) instances (one per core) on each blade server, where each instance allocates no more than a quarter of available RAM (i.e. 3.75GB), as opposed to when executed as a single (1) instance on each blade server which allocates all available RAM (i.e. 15GB). If more than 3.75GB is allocated per instance, the number of cores which can be utilized per blade server (without swapping) is essentially 15GB divided by the amount of RAM allocated per SDK instance (rounded to the nearest whole number).

5.3.2 Multithreaded

An advantage of multithreaded SDKs is the automatic utilization of available processor and memory resources through parallelization (without need for "manual" scheduling). Another advantage is coordinated access (of each thread) to resources such as disk I/O. For this reason NIST strongly recommends that submitted SDKs utilize multithreading aimed at maximizing usage of 4 cores and run in 64-bit mode in order to have access of up to 15GB of RAM.

As a simple example, the execution of a multithreaded SDK for a subtest requiring M latent searches of N exemplars (i.e. Gallery size N), will allocate M searches amongst K available blades such that each blade is executing M/K searches total.

Multithreaded SDKs which run in 64-bit mode have full access to all cores and memory (15GB) on each allocated blade. This approach clearly makes use of processing resources, and has the potential to mitigate contention issues through a coordinated use of parallelism.

Multithreaded SDKs which run in 32-bit mode will be limited to 3GB of RAM per process, which may limit their performance. Another option which exists here is for a multithreaded SDK to use no more than 2 threads, where each SDK instance uses the maximum 3GB of RAM. If informed, NIST could allocate two such SDKs per blade server in order to more fully utilize RAM.

5.4 API

5.4.1 Test Interface Description

Participants shall submit an SDK which provides the interfaces defined in section 5.4.4. Section 5.4.3 defines the interfaces to functions provided by NIST for use by the SDK. Sections 5.4.2 and 5.4.5 specify the declaration of constants, error codes, data-types and functions used by both.

The software undergoing testing will be hosted on NIST-supplied computers. The executable software under test will be built up from two sources: participant-supplied (SDKs) and NIST supplied (image extraction library and test driver).

5.4.2 Declarations

The following are declarations of data types and functions used in the Latent Fingerprint SDK testing interface:

```
////////////////////////////////////////////////////
// Declarations of constants                      //
////////////////////////////////////////////////////

// Impression type codes
#define IMPTYPE_LP      0     // Live-scan plain
#define IMPTYPE_LR      1     // Live-scan rolled
#define IMPTYPE_NP      2     // Nonlive-scan plain
#define IMPTYPE_NR      3     // Nonlive-scan rolled
```

```
// Finger position codes
#define FINGPOS_UK      0     // Unknown finger
#define FINGPOS_RT      1     // Right thumb
#define FINGPOS_RI      2     // Right index finger
#define FINGPOS_RM      3     // Right middle finger
#define FINGPOS_RR      4     // Right ring finger
#define FINGPOS_RL      5     // Right little finger
#define FINGPOS_LT      6     // Left thumb
#define FINGPOS_LI      7     // Left index finger
#define FINGPOS_LM      8     // Left middle finger
#define FINGPOS_LR      9     // Left ring finger
#define FINGPOS_LL      10    // Left little finger


////////////////////////////////////////////////////////////////
// Declarations for the NIST provided library functions       //
////////////////////////////////////////////////////////////////

// Structure to hold a single fingerprint record (image+metadata)
struct finger_record
{
      BYTE    impression_type;
      UINT16  resolution;     // Image resolution in pixels/cm
      BYTE    finger_position;
      UINT16  height;         // Image height in pixels
      UINT16  width;          // Image width in pixels
      BYTE    *image_data;    // 8-bit grayscale image data
};
typedef struct finger_record   FINGER_REC;

// Extracts 10 fingerprint records from a ten-print (AN2K) file
INT32 extract_image_data(     const    char    *tenprint_filename,
                              FINGER_REC **finger_recs);


// De-allocates the memory holding 10 fingerprint records
void free_image_data(FINGER_REC *finger_recs);


////////////////////////////////////////////////////////////////
// Declarations for the SDK provided library functions        //
////////////////////////////////////////////////////////////////


// Extracts features from exemplar
INT32 extract_exemplar( const char *exemplarFilename,
                        const char *outputDir);


// Creates a gallery from set of extracted exemplar features
INT32 create_gallery(   const INT32 numExemplars,
```

```
                        const char **exemplarFeatFilenames,
                        const char *galleryDir);

// Selects the current gallery for latent searching
INT32 set_gallery(      const char *galleryDir);

// Extracts features from latent file
INT32 extract_latent(   const char *latentFilename,
                        const char *outputDir);

// Searches for the latent in the gallery
INT32 latent_search(    const char *latentFeatFilename,
                        const char *outputDir);
```

5.4.3 NIST Provided Functions

5.4.3a Extract Image Data

INT32

```
extract_image_data(const char  *tenprint_filename,
                   FINGER_REC  **finger_recs);
```

Description

This function extracts ten fingerprint image records from a single (AN2K formatted) ten-print record file. The caller shall pass ***tenprint_filename*** as a pointer to the fully qualified pathname of an AN2K formatted ten-print record file, and ***finger_recs*** as the address of a pointer of type `FINGER_REC` (see 5.4.2 above).

Upon return ***finger_recs*** will contain a pointer to an array of ten `FINGER_REC` structures ordered by finger position from 1 (right thumb) to 10 (left little finger). For any fingers that are missing from the original ten-print record file, the *image_data* field in the respective `FINGER_REC` will be a NULL pointer.

Example

```
// Example of processing a ten-print record
FINGER_REC *finger_recs;
INT32 status=extract_image_data("E000123.an2", &finger_recs);
if (status == 0) {
        for (i=0;i<10;i++) {
                if (finger_recs[i].image_data != NULL)
                        process_valid_finger(finger_recs[i]);
                else
                        process_missing_finger(finger_recs[i]);
        }
        free_image_data(finger_recs); // see 5.4.3b below
}
```

Parameters

tenprint_filename (input): A pointer to a ten-print record filename.

finger_recs (output): The address of a FINGER_REC pointer.

Return Value

This function returns *zero* on success or a documented *non-zero* error code otherwise.

5.4.3b Free Image Data

```
void
free_image_data(FINGER_REC *finger_recs);
```

Description

De-allocates all memory used by the array of `FINGER_REC` structures specified by ***finger_recs*** which was allocated during a call to `extract_image_data()`.

Parameters

finger_recs (input): A pointer to an array of FINGER_REC structures.

Return Value

None.

5.4.4 SDK Provided Functions

5.4.4a Exemplar Feature Extraction

```
INT32
extract_exemplar( const char  *exemplarFilename,
                  const char   *outputDir);
```

Description

This function produces a single proprietary formatted feature set file from a 10-print exemplar set. The output from multiple calls to this function (i.e. multiple proprietary feature set files) will be used to construct a gallery (see section 5.4.4b) that is searchable by `latent_search()`.

The 10-print exemplar set will be contained in an ANSI/NIST file with pathname specified by ***exemplarFilename*** (e.g. "/mnt1/input/E1/E199999_1.an2"), and that file will contain either 10 rolled or 10 segmented slap fingerprint images. The directory to which the proprietary feature set file shall be written is specified by the pathname pointed to by ***outputDir*** (e.g. "/mnt/output/feats/E1/").

The format of all pathnames will be canonical Unix style pathnames using forward slash directory separators. The maximum total pathname length is 255 characters.

A single proprietary feature set file shall be written to the directory specified by ***outputDir***. No files other than the feature set file may be written. The filename of the output feature set file is defined here as the base filename of ***exemplarFilename*** with the extension ".an2" replaced by ".feat" (no quotes). For example, if ***exemplarFilename*** ="/mnt1/input/E1/E199999_1.an2" and ***outputDir*** = "/mnt/output/feats/E1/", the proprietary feature set file shall be written as "/mnt/output/feats/E1/E199999_1.feat".

No format is prescribed for the output feature data. For example if desired it may contain images from the 10-print exemplar set. A feature file shall always be output, regardless of any internal failures such as a failure of automated feature extraction. The contents of the directory pointed to by ***outputDir*** (structure and other contents) are not relevant. Pre-computation of feature data avoids reprocessing of the original images upon subsequent calls to `latent_search()`.

The SDK shall use the function `extract_image_data()` (see 5.4.3a above) provided by NIST to extract the raw grayscale image and metadata from the 10-print exemplar set file specified by ***exemplarFilename***. Note that each call to `extract_image_data()` allocates memory to hold the extracted image and metadata, so this memory should be de-allocated using the NIST provided `free_image_data()` (see 5.4.3b above) function when no longer needed.

Return Value

This function returns *zero* on success or a documented *non-zero* error code otherwise.

5.4.4b Gallery Creation

```
INT32
create_gallery(   const INT32  numExemplars,
                  const char   **exemplarFeatFilenames,
                  const char   *galleryDir);
```

Description

This function writes a proprietary enrolled gallery to ***galleryDir*** (e.g. "/mnt/output/gallery/E1/"), based on a list of exemplar feature set file pathnames specified by ***exemplarFeatFilenames***. The gallery shall be usable in read-only mode by subsequent calls to `latent_search()`, and shall associate all exemplar feature sets having the same subject ID (see below). The format of the gallery is at the discretion of the SDK provider. Subdirectories and multiple files may be created within ***galleryDir***. All data produced by the SDK during the execution of this function shall be stored exclusively to the directory specified by ***galleryDir.***

The list of exemplar feature set file pathnames is contained in ***exemplarFeatFilenames***, which is an array of pointers having length ***numExemplars*** + 1, where each element of the array is a pointer to an exemplar feature set file pathname. The last element of the array will be equal to 0 (i.e. a NULL pointer).

The format of all pathnames will be canonical Unix style pathnames using forward slash directory separators. The maximum total pathname length is 255 characters.

Each exemplar feature set file pathname will be formatted *dirPath*"E"*subjectID* "_" *instance* ".feat" (no quotes or spaces), where *dirPath* is the full directory path of the file, *subjectID* is a 6-digit numeric ID (with leading zeros) uniquely identifying the subject, and *instance* is a 1-digit arbitrary numeric index to differentiate between multiple exemplar sets belonging to the same subject. For example, "/mnt/output/feats/E1/E199999_1.feat"

Return Value

This function returns *zero* on success or a documented *non-zero* error code otherwise.

5.4.4c Set Gallery

```
INT32
set_gallery(const char  *galleryDir);
```

Description

This function selects the gallery which shall be used by all subsequent calls to `latent_search()`. The directory pathname specified by ***galleryDir*** (e.g. "/mnt/output/gallery/E1/") shall contain the gallery produced by a prior call to `create_gallery()`.

The format of the pathname will be canonical Unix style pathnames using forward slash directory separators. The maximum total pathname length is 255 characters.

Return Value

This function returns *zero* on success or a documented *non-zero* error code otherwise.

5.4.4d Latent Feature Extraction

```
INT32
extract_latent(   const char *latentFilename,
                  const char *outputDir);
```

Description

This function produces a single proprietary formatted feature set file from an ANSI/NIST file containing a set of 0 or more manually extracted features and a latent fingerprint image (except for subtest LG, see section 7). The proprietary formatted feature set file output by this function will be used as input to `latent_search()`.

The ANSI/NIST file will be specified by a pathname pointed to by ***latentFilename*** (e.g. "/mnt1/input/L3/L12ABC.an2"). The directory to which the proprietary feature set file shall be written is specified by the pathname pointed to by ***outputDir*** (e.g. "/mnt/output/feats/L3/").

The format of all pathnames will be canonical Unix style pathnames using forward slash directory separators. The maximum total pathname length is 255 characters.

A single proprietary formatted feature set file shall be written to the directory specified by ***outputDir***. No format is prescribed for the feature data. The feature data may include any or all manually extracted features already present in the ANSI/NIST file (e.g. it may encode them in a proprietary format). For example if desired it may contain the latent fingerprint image. No files other than the feature set file may be written. A feature file shall always be output, regardless of any internal failures such as a failure of automated feature extraction. The filename of the output feature set file is defined here as the base filename of ***latentFilename*** with the extension ".an2" replaced by ".feat" (no quotes). For example, if ***latentFilename*** = "/mnt1/input/L3/L12ABC.an2" and ***outputDir*** = "/mnt/output/feats/L3/", the proprietary feature set file shall be written as "/mnt/output/feats/L3/L12ABC.feat".

Return Value

This function returns *zero* on success or a documented *non-zero* error code on failure.

5.4.4e Latent Search

```
INT32
latent_search(    const BYTE *latentFeatFilename,
                  const char *outputDir);
```

Description

This function searches the current gallery (as selected by `set_gallery()`) for zero or more candidates matching the input latent feature set (created by `extract_latent()`) whose pathname is specified by ***latentFeatFilename***, and outputs a candidate list to the directory specified by ***outputDir***. The format of the candidate list is specified in section 5.6.

The selection of features on which to match is entirely at the discretion of the SDK. Note that during the call to this function the directory containing the current gallery and its contents are read-only.

The format of all pathnames will be canonical Unix style pathnames using forward slash directory separators. The maximum total pathname length is 255 characters.

One candidate list file (per call to this function) shall be written to the directory specified by ***outputDir***. A candidate list file shall always be output, regardless of any internal software failures. The filename of the candidate list file is defined here as the base filename of ***latentFeatFilename*** with the extension ".feat" replaced by ".CL" (no quotes). For example, if ***latentFeatFilename*** = "/mnt1/output/feats/L3/L12ABC.feat" and ***outputDir*** = "/mnt/output/clists/L3/", the candidate list file shall be written as "/mnt/output/clists/L3/L12ABC.CL".

Note 1: Since it may not be possible to keep all gallery data in memory, it might be necessary for the software to repeatedly retrieve the data from disk, and this extra fetch time will be included in the execution time measurement.

Note 2: The candidate list shall only depend on the inputs to this function and the currently selected gallery (not on any previous results from this function). Thus, identical latent feature inputs and gallery data shall produce identical candidate lists independent of all prior calls to this function.

Return Value

This function returns *zero* on success or a documented *non-zero* error code on failure.

5.4.5 Error Codes and Handling

The participant shall provide documentation of all (non-zero) error or warning return codes (see section 5.4.8, Documentation).

The application should include error/exception handling so that in the case of a fatal error, the return code is still provided to the calling application.

All messages which convey errors, warnings or other information shall be suppressed, except where they may provide additional information not conveyable by the defined error codes alone (such as listing a specific file related to the error).

At minimum the following return codes shall be used.

Return code	Function	Explanation
0	All	Success
-1	extract_image_data()	unable to open file
-2	extract_image_data()	Incorrect file format
-3	extract_image_data()	error parsing ten-print file
-4	extract_image_data()	error decompressing image
-5	extract_image_data()	insufficient memory error
-6	extract_image_data()	unspecified error
100	extract_exemplar()	exemplar file not found
101	extract_exemplar()	output directory not found
102	extract_exemplar()	unable to write feature data

103	extract_exemplar()	error from extract_image_data (write to stdout)
201	create_gallery()	feature file not found (write filename to stdout)
202	create_gallery()	output directory not found
203	create_gallery()	unable to write gallery enrollment data
204	create_gallery()	insufficient memory available
301	extract_latent ()	latent file not found
302	extract_latent()	output directory not found
303	extract_latent()	unable to write feature data
401	set_gallery()	gallery directory not found
501	latent_search()	gallery directory not set
502	latent_search()	insufficient memory available
503	latent_search()	feature file not found
504	latent_search()	candidate list directory not found
505	latent_search()	unable to write candidate list

5.4.6 SDK Library and Platform Requirements

Participants shall provide NIST with binary code only (i.e. no source code) – supporting files such as header (".h") files notwithstanding.

Note that dependencies on external dynamic/shared libraries such as compiler-specific development environment libraries are discouraged. If absolutely necessary, external libraries must be provided to NIST upon prior approval by the Test Liaison.

The SDK will be tested in non-interactive "batch" mode (i.e. without terminal support). Thus, the library code provided shall not use any interactive functions such as graphical user interface (GUI) calls, or any other calls which require terminal interaction.

The use of multi-threading by the SDK is encouraged as the NIST test platform includes dual-processor dual-core support. The SDK need not be "thread safe" as the NIST test driver itself is single threaded. If multi-threading is utilized by the SDK is shall be documented.

NIST will link the provided library file(s) to a C language test driver application (developed by NIST) using the GCC compiler (*for Windows platforms Cygwin/GCC version 3.3.1 will be used; for Linux platforms GCC version 4.1.2 and GNU ld 2.17.50.0.6-5.el5 will be used. All GCC compilers use Libc 6*). For example,

```
gcc  –o latenttest  latenttest.c  -L. –lelftEfsSDK
```

Participants are required to provide their library in a format that is linkable using GCC with the NIST test driver, which is compiled with GCC. All compilation and testing will be performed on x86 platforms running either Windows 2008 Server or Red Hat Enterprise Linux Server release 5.1 "Tikanga" (kernel 2.6.18-53 or higher) dependent upon the operating system requirements of the SDK. Thus, participants are strongly advised to verify library-level compatibility with GCC (on an equivalent platform) prior to submitting their software to NIST to avoid linkage problems later on (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.).

5.4.7 Installation and Usage

The SDK must install easily (i.e. one installation step with no participant interaction required) to be tested, and shall be executable on any number of machines without requiring additional machine-specific license control procedures or activation.

The SDK's usage shall be unlimited. No usage controls or limits based on licenses, execution date/time, number of executions, etc. shall be enforced by the SDK.

It is requested that the SDK be installable using simple file copy methods, and not require the use of a separate installation program. Contact the Test Liaison for prior approval if an installation program is absolutely necessary.

5.4.8 Documentation

Complete documentation of the SDK shall be provided, and shall detail any additional functionality or behavior beyond what is specified in this document. The documentation must define all error and warning codes.

5.5 Software execution process

The execution process will take place in three passes:

- Exemplar feature extractions and Gallery creation
- Latent image feature extractions
- Latent searches against each Gallery

5.6 Format of Candidate List

The result of the latent_search() function is a candidate list, saved as a tab-delimited text file. The candidate list has a fixed length of one hundred (100) candidates. The candidate list consists of two parts, a required and an optional part.

The required part consists of:

- the ID of the mating exemplar subject
- the matching finger number
- the absolute matching score
- an estimate of the probability of a match (0 to 100)

The optional part consists of:

- the number of minutiae identified in the latent
- the number of latent minutiae which were successfully matched

Sample Candidate List						
Required Part					Optional Part	
Rank	Mate ID	Finger No.	Abs. Score	Prob. Of True Match	No. Latent Minutiae	Minutiae Matched
1	073141	2	3513	93	18	12
2	199999	2	605	5	18	5
3	004334	3	513	4	18	5
…						
100	920792	9	422	1	18	4

Table 1: Sample candidate list

The candidate list is ordered based upon the absolute score, with the highest score in the first position.

The parameter *Probability of True Match* is an estimate of the probability that the candidate is a true match. Its values range from 0 to 100.

Each candidate list will be stored in an individual tab-delimited ASCII text file. Within the candidate list file, all required and optional parts for an individual candidate entry (i.e. row) should be written one per-line in the order shown above, with each part (i.e. column) separated by a single tab character.

Note that "Mate ID" shall be written as the 6-digit *subjectID* (see section 5.4.4b) part of the exemplar filename specified to the create_gallery() function. E.g. if "/mnt/output/feats/E1/E199999_1.feat" was enrolled to the gallery being searched, the Mate ID shall be "199999", without quotes). Note also that the candidate list refers to a subject and finger position, not a specific exemplar impression.

5.7 Validation

As discussed in Section 3.1, a Validation Dataset will be provided to verify the correct operation of participants' software before and after delivery to NIST. Using this data and the submitted SDK, identical outputs must be generated by NIST to those submitted by participants in order for the submitted SDK to be accepted. Acceptance of the submitted SDK must occur prior to the deadlines specified in section 6.

The Validation Dataset will be a small subset of the ELFT-EFS Public challenge dataset.

5.8 Timing Requirements

The ELFT-EFS Evaluation test must place limits on the processing time of the major operations involving feature extraction and enrollment (exemplars and latents) and searching. There are two purposes for such limits. The first is to enable practical execution of the test within an acceptable period of time. The second is to measure performance at throughput rates comparable to large-scale operational scenarios. Our sponsors have interest in relevance of results to near-term operational requirements. The size of the test will be dictated to a large extent by these throughput numbers.

SDK time limits are specified on a "per-core" basis, meaning that the specified operational rates are for a single core – in other words, rates will be specified from the perspective of a sequential process executing on a single CPU core. For example, if the specified rate for latent search is R exemplars per second, then a multithreaded SDK instance operating on 4 cores must achieve an aggregate rate of 4 x R. All time limits below are averages with respect to the hardware used on the NIST test platform specified above.

The search time requirements specified below are for Subtests LC-LG: see Section 7 for details. It is recognized that for some implementations, throughput for image□only searches (Subtest LA) may be slower due to less effective screening. It is allowable for throughput on Subtest LA (image only) and LB (image+ROI) to be slower by a factor of up to 2x than the stated search time.

Proposed time limits for the ELFT-EFS Evaluation are (per single CPU core):

Exemplar feature extraction	100 sec/10-finger exemplar set (rolled or pre-segmented slap)
Latent enroll	120 sec/latent
Search	0.025 sec/10-finger exemplar set Rate of 40 exemplar sets/sec, per latent (exemplar set = 10 all rolled or all plain prints)

Table 2: Timing requirements

6 Schedule and Software Submission Requirements

To enable enrolling the gallery before the evaluation itself takes place, we are requesting the exemplar feature extraction/enrollment SDKs prior to the latent feature extraction/search SDKs. For each SDK, we have both early and final deadlines: we will accept SDKs as early as the early deadline, and will use the period from receipt of the SDKs until the final deadline to validate correct operation of the SDKs, but must have fully operational software by the final deadline. Between the early and final deadlines, we will report any software issues encountered, and will accept software replacements.

If major software problems arise during the execution of the evaluation (i.e. after the submission deadline), reasonable attempts will be made to resolve the issue(s) through reporting and receipt of replacement software. However replacement software must not include algorithm enhancements beyond those addressing the specific problem(s) reported.

Registration/Withdraw

- Registration form online: 12 April 2010
- Registration deadline: 15 May 2010
- Deadline for anonymous withdraw: 21 June 2010

Exemplar feature extraction / enrollment SDKs:

- Submission Period Open: 15 May 2010
- Final deadline: 21 June 2010

Latent feature extraction / search SDKs:

- Submission Period Open: 15 May 2010
- Final deadline: 21 June 2010

7 EFS Fields Used

			Subtest combinations for ELFT-EFS Evaluation 1						
Abb.	#	Field Name	Subtest LA: Image only	Subtest LB: ROI	Subtest LC: ROI, Pattern Class, Quality Map	Subtest LD: IAFIS/ EFTS equivalent	Subtest LE: Baseline EFS	Subtest LF: Baseline EFS with Skeleton	Subtest LG: IAFIS/ EFTS equivalent
			With Image						Without Image
LEN	9.001	Logical Record Length	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IDC	9.002	Image Designation Character	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IMP	9.003	Impression Type	Yes	Yes	Yes	Yes	Yes	Yes	Yes
FMT	9.004	Minutiae Format	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ROI	9.300	Region of Interest		Yes	Yes	Yes	Yes	Yes	Yes
ORT	9.301	Orientation		Yes	Yes	Yes	Yes	Yes	Yes
FPP	9.302	Finger/Palm Position(s)							
PAT	9.307	Pattern Classification			Yes	Yes (**)	Yes	Yes	Yes (**)
RQM	9.308	Ridge Quality Map			Yes		Yes	Yes	
RQF	9.309	Ridge Quality Map Format			Yes		Yes	Yes	
RFM	9.310	Ridge Flow Map						Yes	
RFF	9.311	Ridge Flow Map Format						Yes	
RWM	9.312	Ridge Wavelength Map							
RWF	9.313	Ridge Wavelength Map Format							
TRV	9.314	Tonal Reversal		Yes	Yes	Yes	Yes	Yes	Yes
PLR	9.315	Possible Lateral Reversal							
FQM	9.316	Friction Ridge Quality Metric							
PGS	9.317	Possible Growth or Shrinkage							
COR	9.320	Cores				Yes	Yes	Yes	Yes

DEL	9.321	Deltas				Yes	Yes	Yes	Yes
CDR	9.322	Core-Delta Ridge Counts				Yes	Yes	Yes	Yes
CPR	9.323	Center Point of Reference					Yes	Yes	
DIS	9.324	Distinctive Characteristics					Yes	Yes	
NCR	9.325	No Cores Present					Yes	Yes	
NDL	9.326	No Deltas Present					Yes	Yes	
NDC	9.327	No Distinctive Areas Present					Yes	Yes	
MIN	9.331	Minutiae				Yes (*)	Yes	Yes	Yes (*)
MRA	9.332	Minutiae Ridge Count Algorithm							
MRC	9.333	Minutiae Ridge Counts				Yes	Yes	Yes	Yes
NMP	9.334	No Minutiae Present					Yes	Yes	
RCC	9.335	Ridge Count Confidence					Yes	Yes	
DOT	9.340	Dots					Yes	Yes	
INR	9.341	Incipient Ridges					Yes	Yes	
CLD	9.342	Creases and Linear Discontinuities					Yes	Yes	
REF	9.343	Ridge Edge Features					Yes	Yes	
NPP	9.344	No Pores Present					Yes	Yes	
POR	9.345	Pores					Yes	Yes	
NDT	9.346	No Dots Present					Yes	Yes	
NIR	9.347	No Incipient Ridges Present					Yes	Yes	
NCR	9.348	No Creases Present					Yes	Yes	
NRE	9.349	No Ridge Edges Present					Yes	Yes	
MFD	9.350	Method of Feature Detection							
COM	9.351	Comments							
LPM	9.352	Latent Processing Method							
EAA	9.353	Examiner Analysis Assessment					Yes	Yes	
EOF	9.354	Evidence of Fraud							
LSB	9.355	Latent Substrate							
LMT	9.356	Latent Matrix							
LQI	9.357	Local quality issues					Yes	Yes	
AOC	9.360	Area of Correspondence							
CPF	9.361	Corresponding Points or Features							
ECD	9.362	Examiner Comparison Determination							
SIM	9.372	Skeletonized Image						Yes (***)	
RPS	9.373	Ridge Path Segments							