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.

A-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.

A-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.

A-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.

A-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.

A-5.4.5 Error Codes and Handling

The participant shall provide documentation of all (non-zero) error or warning return codes (see section A-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

A-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.).

A-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.

A-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.

A-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

A-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.

A-5.7 Validation

As discussed in Section A-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.

A-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 A-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

A-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: 13 July 2009
- Registration deadline: 27 July 2009
- Deadline for anonymous withdraw: 16 August 2009

Exemplar feature extraction / enrollment SDKs:

- Early deadline: 2 August 2009
- Final deadline: 16 August 2009
- Preparation of galleries will start when SDKs are validated, but no later than Monday 17 August

Latent feature extraction / search SDKs:

- Early deadline: 16 August 2009
- Final deadline: 30 August 2009
- Latent evaluations to start post SDK validation, but no later than Monday 31 August

A-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							