Minutiae Interoperability Exchange Test 2004 (MINEX04)

API Specification

Overview

The Minutiae Interoperability Exchange Test 2004 (MINEX04) will determine the feasibility of using minutiae data (rather than image data) as the interchange medium for fingerprint information between different fingerprint matching systems.

All images selected for use in MINEX04 have been gathered from subjects using live-scan devices or traditional paper & ink methods. No latent fingerprint images will be used. Furthermore all testing of fingerprint minutiae based identification systems in MINEX04 will be limited to automated, software-based systems.

Particpants in MINEX04 shall be required to provide NIST with an SDK library which complies with the Application Program Interface (API) specified in this document. The SDK library provided shall run on Pentium-based PC platforms using Windows 2000 or Red Hat Linux 7.2.

The SDK provided must include functionality to extract a set of minutiae data from an individual fingerprint image and compute a match-score developed by comparing one set of minutiae data with another. Participants submitting an SDK for testing shall at a minimum provide support for minutiae extraction and minutiae template matching based on (1) the participant's proprietary *non-standardized* minutiae extraction method and/or template format (used to provide a baseline measurement and not designed for the interoperability tests); and (2) the *basic* (i.e. without extended data) ANSI INCITS 378-2004 [1] minutiae data format standard (as implemented by this document).

Furthermore, in order to determine if significantly improved interoperability can be achieved with the use of ridge crossing information (in addition to the basic location and angle information), participants are strongly encouraged and requested to provide additional support for ANSI INCITS 378-2004 templates (as implemented by this document) *with extended data* including ridge count information for the 8 neighbor-octants surrounding each minutia. If ridge count information is provided, participants may optionally include Core and Delta information in the extended data area as well.

1 Fingerprint Image Data

1.1 *Format*

The SDK must be capable of processing fingerprint images supplied to the SDK in uncompressed raw 8-bit (one byte per pixel) grayscale format. Each image shall appear to have been captured in an upright position and approximately centered horizontally in the field of view. The image data shall appear to be the result of a scanning of a conventional inked impression of a fingerprint. Figure 1 illustrates the recording order for the scanned image. The origin is the upper left corner of the image. The x-coordinate (horizontal) position shall increase positively from the origin to the right side of the image. The y-coordinate (vertical) position shall increase positively from the origin to the bottom of the image.

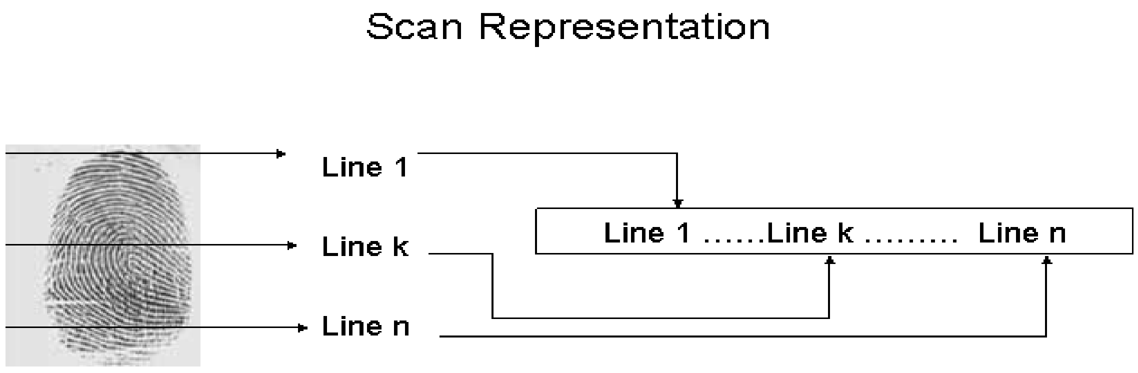

Figure 1 Order of scanned lines

Raw 8-bit grayscale images are canonically encoded. The minimum value that will be assigned to a "black" pixel is zero. The maximum value that will be assigned to a "white" pixel is 255. Intermediate gray levels will have assigned values of 1- 254. The pixels are stored left to right, top to bottom, with one 8-bit byte per pixel. The number of bytes in an image is equal to its height multiplied by its width as measured in pixels; there is no header. The image height and width in pixels will be supplied to the SDK as supplemental information.

1.2 *Resolution and Dimensions*

All images for this test will employ 500 PPI resolution (horizontal and vertical).

The dimensions of the fingerprint images will vary from 150 to 812 pixels in width, and 166 to 1000 pixels in height.

Note – the SDK must be capable of processing images with any dimensions in these specified ranges without the use of separately invoked cropping or padding facilities. For example, SDKs which require cropping of large images must do so internal to the operation of the create_template (see below) API call.

1.3 *Sensor and Impression Types*

The images used for this test come from a variety of sensors, and include both live-scanned and nonlive-scanned rolled and plain impression types.

These sensors will include but not be restricted to Identix DFR-90, Crossmatch 300A, Smith-Heimann V300, and scanned paper.

2 Minutiae Data

2.1 Minutiae Identification and Placement

For each participant's non-standardized (e.g. proprietary) template implementation(s), all details regarding the identification, placement (determination of location and angular direction), and encoding of individual minutiae shall be according to the participant, and are beyond the scope of this document.

For templates compliant with the basic ANSI INCITS 378-2004 (as implemented by this document), identification and placement (determination of location and angular direction) of individual minutiae shall be according to section 5 of ANSI INCITS 378-2004, and shall be limited to minutia of type "ridge ending" and "ridge bifurcation".

For templates compliant with ANSI INCITS 378-2004 (as implemented by this document) ***with*** an extended data area (i.e. containing ridge count information), the identification and placement (determination of location and angular direction) of individual minutiae shall follow the approach discussed sections 2.1.1 and 2.1.2 of this document, and shall be limited to minutia of type "ridge ending" and "ridge bifurcation". This approach, which enhances ANSI INCITS 378-2004, is based on the techniques used by the FBI's IAFIS system, and provides guidelines for identifying minutiae and determining their placement. Note that this approach, while differing from the one described in section 5 of ANSI INCITS 378-2004, does not require the extraction of any information beyond what is specified by that standard (i.e. only the *method* of identification and placement differs from ANSI INCITS 378-2004, the same kind of information is extracted).

2.1.1 Minutiae Location

Before the minutiae can be identified, the fingerprint is usually first binarized. The resultant image is then used as input into a skeletonization process that simultaneously thins the image until both the ridges and valleys are represented by one-pixel thick sequences of pixels. This process creates a minutiae representation of the image consisting only of bifurcations. Ridge bifurcations are identified and located directly by the process, while ridge endings have been effectively converted into valley skeleton bifurcations (as a result of thinning the valleys). This process can be considered as an implementation of the medial skeleton approach used by ANSI INCITS 378-2004. At this point, all of the minutiae are represented as bifurcations. The X and Y pixel coordinates of the intersection of the three legs of each minutia can be directly recorded in the template.

2.1.2 Minutiae Direction

Determination of the minutia direction can be extracted from each skeleton bifurcation. The three legs of every skeleton bifurcation must be examined and the endpoint of each leg determined. Figures 2A through 2C illustrate the three methods used for determining the end of a leg. The ending is established according to the event that occurs first.

- The 32nd pixel – Figure 2A and 2B
- The end of skeleton leg if greater than 10 pixels (legs shorter are not used) – Figure 2A
- A 2nd bifurcation is encountered before the 32nd pixel – Figure 2C

The angle of the minutiae is determined by constructing three virtual rays originating at the bifurcation point and extending to the end of each leg. The smallest of the three angles formed by the rays is bisected to indicate the minutiae direction.

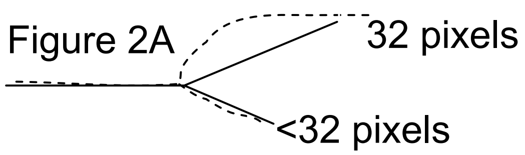

Figure 2A

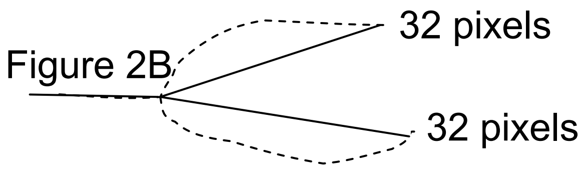

Figure 2B

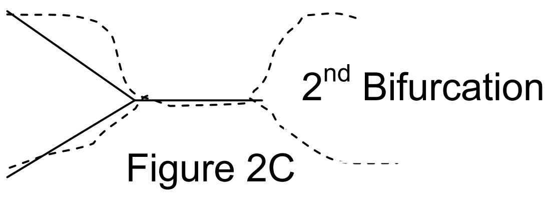

Figure 2C

2.2 Minutiae Type

ANSI INCITS 378-2004 requires that each minutia have a type associated with it which is stored in the template. Templates compliant with ANSI INCITS 378-2004 (as implemented by this document) shall be limited to minutiae of type "ridge ending" and "ridge bifurcation”. Minutiae not satisfying these definitions shall not be extracted or included in the basic or extended data templates. For those cases where it is not possible to reliably distinguish between a ridge ending and a bifurcation, the category of "other" shall be used. This is a common characteristic of "inked" impressions that exhibit ridge endings being converted to bifurcations and bifurcations being converted to ridge ending due to over- or under-inking found in the image. Vendors whose systems do not distinguish between minutiae type or do not rely on minutiae type in their extraction or matching algorithms may assign the "other" type to all minutiae.

Note – Although ANSI INCITS 378-2004 uses the category of "other" to describe minutiae that are neither ridge endings nor ridge bifurcations, such as trifurcations and crossovers, MINEX04 has no provision for minutiae types other than ridge endings and bifurcations.

2.3 Minutiae Coordinate System and Angle Conventions

For templates compliant with ANSI INCITS 378-2004 (as implemented by this document), the coordinate system and angle conventions used shall be those defined in sections 5.3.1 and 5.4.1, respectively, of ANSI INCITS 378-2004.

Note that for MINEX04 it is additionally required that all coordinates and angles for minutiae shall be recorded with respect to the *original* image input to the SDK. In other words, they shall not be recorded with respect to any image processing subimage(s) created by the SDK during the minutiae extraction process.

2.4 ANSI INCITS 378-2004 Minutiae Data Format

The SDK must support the creation and matching of fingerprint minutiae templates in the format specified by both ANSI INCITS 378-2004 "Fingerprint Minutiae Format for Data Interchange" and this document.

At a minimum, the SDK shall support ANSI INCITS 378-2004 compliant templates (as implemented by this document) ***without*** extended data.

Support for ANSI INCITS 378-2004 compliant templates (as implemented by this document) ***with*** extended data is strongly encouraged but optional.

2.4.1 Constraints

A template that is created and/or processed by the SDK which complies with ANSI INCITS 378-2004 shall be formatted as a non-encapsulated byte-array containing an ANSI INCITS 378-2004 minutiae record and with the following constraints:

Note – templates which fail to meet any of the conditions below will be considered non-standardized, and will be tested as such (i.e. they will not be tested as ANSI INCITS 378-2004 type templates).

- CBEFF wrappers shall not be used.
- The record header shall be fixed at 26 bytes in length (i.e. only 2 bytes shall be used to record the total record length in the header).
- The CBEFF PID (Product ID) field shall be set to 0.
- The number of Finger Views shall be limited to one.

- The fields *Finger Position*, *Impression Type*, and *Size of Scanned Image* (in both the x and y directions) shall be input by the test application at run-time.
- The field *Impression Type* shall only range in values from 0 through 3 that will include plain and rolled, live-scan and nonlive-scan images.
- The field *Capture Equipment ID* shall be set to 0 (unreported).
- The field *Capture Equipment Compliance* shall be set to 0.
- The *Minutia Quality* field for each minutia shall be set to 0.
- The fields *X (horizontal) resolution* and *Y (vertical) resolution* shall be set to 197 (pixels/cm).
- The maximum number of minutiae encoded in a template shall be 128.
- The field *Minutiae Type* shall be set in accordance with ANSI INCITS 378-2004 section 6.5.2.1 with the exception of type "other" which shall be assigned in accordance with section 2.2 of this specification, thus:
    - 01 – ridge ending
    - 10 – ridge bifurcation
    - 00 – ridge ending or bifurcation (but undetermined as to which)
- Extended data shall be optional (i.e. it is not required).
- Extended data shall not include vendor-defined information.
- Extended data, if supplied, must include Ridge Count information extracted using the *Eight-neighbor Ridge Count Extraction Method* (as defined in section 2.4.2 below). Nonspecific or four-neighbor extraction methods are *not* permitted.
- Extended data may include Core and Delta information, but only in addition to Ridge Count information (i.e. Core and Delta information without Ridge Counts are not permitted).
- Core and Delta information shall be limited to a maximum of two Cores and two Deltas.
- Minutiae records shall not exceed 4500 bytes in length.
- The field *Finger Quality* will be input by the test application (and shall be output identically by the SDK) at run-time and represent only the quality of the original image. It will be limited to 5 discrete values, which directly correspond (as shown in Table 1 below) to values on the NIST Fingerprint Image Quality (NFIQ) [3] scale. This image quality information may be used by each SDK to alter its normal processing paths within the match function.

Finger Quality value used in MINEX	Description	NIST NFIQ Value (for reference only)
1	Poor	5
25	Fair	4
50	Good	3
75	Very Good	2
100	Excellent	1

Table 1 - Finger Quality values

Integer values other than those shown in Table 1 are not defined and will not be provided by the test application.

2.4.2 Eight-neighbor Ridge Count Extraction Method

Templates containing extended data shall contain ridge count information extracted for *every* minutia recorded in the minutiae data area (refer to section 6.5.2 of ANSI INCITS 378-2004) of the template. Ridge count information shall be extracted as follows.

- Every minutia identified in the minutiae data area shall be assigned its own unique "neighborhood" consisting of eight octants (angular sectors of 45 degrees) of a (theoretical) circle centered on the location of the minutia. The octants shall be numbered counterclockwise from zero to seven with octant number zero locally center aligned with the direction of the minutiae. Figure 3 provides an example of a center minutiae whose "tail" is aligned toward the "South-Southwest" direction. The zero octant will span the arc of 22.5 degrees on either side of "South-Southwest". The "tail" bisects the zero octant.

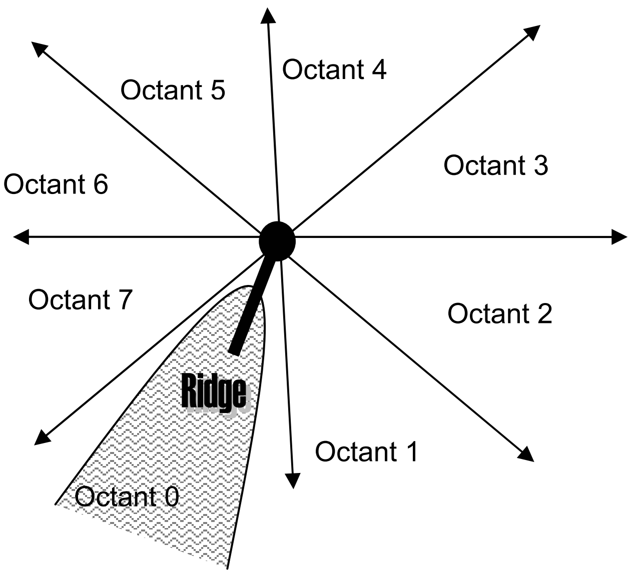

Figure 3 – Minutiae neighborhood

- For each octant, a ridge count is produced by counting the number of ridges crossed by a (theoretical) straight line *between* the center minutia and the minutia *nearest* to it (i.e. its "nearest neighbor") in that octant, *including* the ridge on which the nearest neighbor minutia lies (i.e. the number of intervening ridges plus one).

Valid ridge counts range from zero to 15. If a given octant has no neighboring minutiae in it, the ridge count shall be zero. If the number of ridges counted exceeds 13, the ridge count shall be set to14. If the count cannot be calculated because the intervening image area between the two minutiae is blurred and insufficient to estimate an accurate ridge count, the count shall be set to 15.

Note that the ridge(s) defining the minutia are considered part of the minutia. For example, if the center minutia is a bifurcation, and if the straight line passes through one of the ridge segments forming the bifurcation, the latter is not counted separate from the minutia and does not increment the count.

Note also that due to the curving of ridges, the straight line between a minutia and its neighbor might cross a ridge twice (or more). In this case the same ridge shall not be counted more than once, unless the straight line which connects the center minutia to its nearest neighbor minutia passes through at least the center of one valley before entering back into the same ridge. A ridge can not be counted unless the straight line passes at least to the center of the ridge thickness.

Minutia Index	**Minutia Neighbor Index**	**Ridge Count**	*Octant Index (not stored)*
0x01	0x06	0x05	*0*
0x01	0x03	0x09	*1*
0x01	0x00	0x00	*2*
…	…	…	…
0x01	0x11	0x0D	*7*
0x02	0x04	0x05	*0*
0x02	0x10	0x09	*1*
…	…	…	…
0x80	0x6F	0x02	*7*

Table 2 – Example Ridge Count Data

The formatting and storage of Ridge Count data in the extended data section shall be according to ANSI INCITS 378-2004. ANSI INCITS 378-2004 specifies that all ridge count data shall be stored in increasing order of (center) minutia index number, with 8 neighbor-octant counts recorded for each. In contrast to ANSI INCITS 378-2004 however, this specification requires that ridge counts be recorded in sequential octant index order for each center minutiae. Octants with no neighboring minutiae shall be recorded with the minutia neighbor index and the ridge count set to 0. Table 2 is an example of the proper ordering for ridge count information.

2.5 *Non-Standardized Minutiae Data Format*

A template that is created or processed by the SDK in a manner or format which does not comply with ANSI INCITS 378-2004 (as implemented by this document) – for example proprietary templates – will be considered non-standardized. The specific creation method or format of non-standardized templates shall not be disclosed by the Participant, and all such templates will be treated as opaque data objects in MINEX04. The SDK must support at least one such non-standardized minutiae extraction and/or template format.

Note that the length of non-standardized templates may vary, however it is strongly recommended that their length not exceed 8192 bytes (8K) in length.

3 Testing Interface Description

MINEX04 participants shall submit an SDK which provides the following interface (shown in C-style ***pseudo-code*** prototypes).

3.1 Pre-defined Values

The following are pre-defined values (constants) for use in specifying parameters to the MINEX04 testing interface:

```
// Template type codes

#define MIN_A       0x0100  // Basic M1 (x,y,theta)
#define MIN_B       0x0101  // Basic M1 + ridge counts
```

Note: `MIN_B` may optionally include Core & Delta information in addition to the Ridge Count Information.

```
#define MIN_PROP     0x0200  // LSB indexes format
```

Note: The least-significant byte (LSB) of `MIN_PROP` is used for indexing specific proprietary template formats where more than one such format is generated and/or matched by an SDK. It is the responsibility of those submitting their SDK toMINEX04 to document all such formats in so far as the index value to use for selecting each one.

```
// Impression type codes

#define IMPTYPE_LP  0x00     // Live-scan plain
#define IMPTYPE_LR  0x01     // Live-scan rolled
#define IMPTYPE_NP  0x02     // Nonlive-scan plain
#define IMPTYPE_NR  0x03     // Nonlive-scan rolled

// Finger position codes

#define FINGPOS_UK  0x00     // Unknown finger
#define FINGPOS_RT  0x01     // Right thumb
#define FINGPOS_RI  0x02     // Right index finger
#define FINGPOS_RM  0x03     // Right middle finger
#define FINGPOS_RR  0x04     // Right ring finger
#define FINGPOS_RL  0x05     // Right little finger
#define FINGPOS_LT  0x06     // Left thumb
#define FINGPOS_LI  0x07     // Left index finger
#define FINGPOS_LM  0x08     // Left middle finger
#define FINGPOS_LR  0x09     // Left ring finger
#define FINGPOS_LL  0x0A     // Left little finger

// Finger quality values

#define QUAL_POOR       1    // NFIQ value 5
#define QUAL_FAIR      25    // NFIQ value 4
#define QUAL_GOOD      50    // NFIQ value 3
#define QUAL_VGOOD     75    // NFIQ value 2
#define QUAL_EXCELLENT 100   // NFIQ value 1
```

3.2 Minutiae Extraction and Matching

3.2.1 Get Template Size

```
INT32
get_max_template_size(const UINT16 template_type,
                      const UINT16 height,
                      const UINT16 width,
                      UINT *size);
```

Description

This function returns the maximum number of bytes required for the ANSI INCITS 378-2004 or non-standardized (e.g. proprietary) template as specified by the ***template_type*** parameter, given the ***height*** and ***width*** dimensions (in pixels) of an input raw image. It may be assumed that memory for the ***size*** parameter is allocated before the call.

Parameters

`template_type` **(input)**: The type of template indicated (e.g. MIN_B).

`height` **(input)**: The number of pixels indicating the height of the image.

`width` **(input)**: The number of pixels indicating the width of the image.

`size` **(output)**: The maximum template size in bytes.

Return Value

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

3.2.2 Create Template

```
INT32
create_template(const BYTE* raw_image,
       const BYTE   image_quality,
       const UINT16 template_type,
       const BYTE   finger_position,
       const BYTE   impression_type,
       const UINT16 height,
       const UINT16 width,
       BYTE *template);
```

Description

This function takes a raw image as input and outputs the corresponding standardized or proprietary template as specified by the ***template_type*** parameter. The memory for the template is allocated before the call (i.e., `create_template()` does not handle the memory allocation for the ***template*** parameter). The function returns either success (0) or failure (non-zero). Failure indicates a failure to enroll the image and will result in the output of a *null template* which may be used in later comparisons.

Note – For templates of type MIN_A and MIN_B, *null templates* are defined as having an ANSI INCITS 378-2004 compliant header (only) with the fields *View Number* and *Number of Minutiae* set to 0. No data follows the header in this case.

Parameters

`raw_image` **`(input)`**: The uncompressed raw image used for template creation.

`image_quality` **`(input)`**: The quality of the image (e.g. QUAL_EXCELLENT).

`template_type` **`(input)`**: The template type to be output (e.g. MIN_B).

`finger_position` **`(input)`**: The finger position code (e.g. FINGPOS_LI).

`impression_type` **`(input)`**: The impression type code (e.g. IMPTYPE_LP).

`height` **(input)**: The number of pixels indicating the height of the image.

`width` **(input)**: The number of pixels indicating the width of the image**.**

`template` **(output)**: The processed template.

Return Value

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

3.2.3 Match Templates

```
INT32
match_templates(const UINT16 probe_template_type,
        const BYTE*  probe_template,
        const UINT16 gallery_template_type,
        const BYTE*  gallery_template,
        float* score);
```

Description

This function compares two templates and outputs a match score. The ***probe_template*** parameter shall be compared to the ***gallery_template*** parameter (in this order, if the SDK's matching operation is order dependent). The score returned is a floating-point number which represents the *similarity* of the original fingerprint images from which the templates where created. Scores should not be quantized. It may be assumed that memory for the ***score*** parameter is allocated before the call. **Note that comparisons in which either template is a *null template* (see 3.2.2 above) shall cause the matching operation to fail and output a (documented) error code.**

Parameters

`probe_template_type` **(input)**: The type of ***probe template*** (e.g. MIN_A).

`probe_template` **(input)**: A template returned by `create_template()`.

`gallery_template_type` **(input)**: The type of ***gallery_template*** (e.g. MIN_A)

`gallery_template` **(input)**: A template returned by `create_template()`.

`score` **(output)**: A similarity score resulting from comparison of the templates.

Return Value

This function returns *zero* on success (i.e. a valid score was produced) or a documented *non-zero* error code on failure. In the latter case, the function is expected to return a score which is outside the range of legitimate score values, for example -1.

3.3 Error Codes and Handling

The participant shall provide documentation of all (non-zero) error or warning return codes (see section 4.3, 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.

At minimum the following return codes shall be used.

Return	Explanation
0	Success
1	Image size not supported
2	Template type not supported
3	Failed to extract minutiae (template creation failed)

4	Failed to match templates – *null* probe or gallery template
5	Failed to match templates – unable to parse probe template
6	Failed to match templates – unable to parse gallery template

All messages which convey errors, warnings or other information shall be suppressed.

4 Software and Documentation

4.1 SDK Library and Platform Requirements

Individual SDKs provided must not include multiple "modes" of operation, or algorithm variations. No switches or options will be tolerated within one library. For example, the use of 2 different "coders" by a minutiae extractor must be split across 2 separate SDK libraries.

Participants shall provide NIST with binary code only (i.e. no source code) − supporting files such as header (".h") files notwithstanding. It is preferred that the SDK be submitted in the form of a single static library file (ie. ".LIB" for Windows or ".a" for Linux). However, dynamic/shared library files are permitted.

If dynamic/shared library files are submitted, it is preferred that the API interface specified by this document be implemented in a single "core" library file with the base filename 'libminex' (for example, 'libminex.dll' for Windows or 'libminex.so' for Linux). Additional dynamic/shared library files may be submitted that support this "core" library file (i.e. the "core" library file may have dependencies implemented in these other libraries).

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 (e.g. calls to "standard input" or "standard output").

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 3.3.3 will be used; for RedHat Linux 7.2 platforms GCC version 2.96 will be used. All GCC compilers use Libc 6*). For example,

```
gcc  −o mintest  mintest.c  -L. −lminex
```

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 2000 or Red Hat Linux 7.2 (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.).

4.2 Installation

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.

It is recommended 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.

4.3 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.

4.4 Speed

On average, a template match operation shall take no more than 10 milliseconds, and a template creation operation shall take no more than 1 second to complete (using a 2GHz Pentium IV).

Processing speed will be noted but will not be a primary evaluation criterion.

References

[1] *American National Standard for Information Technology - Finger Minutiae Format for Data Interchange*, ANSI/INCITS 378-2004, www.incits.org

[2] R. M. McCabe, "ANSI/NIST-ITL 1-2000 Data Format for the Interchange of Fingerprint, Facial, and Scar Mark & Tattoo (SMT) Information," ftp://sequoyah.nist.gov/pub/nist_internal_reports/sp500-245-a16.pdf

[3] E. Tabassi, et al. “Finger Print Image Quality,” NISTIR 7151 2004 (Gaithersburg, MD: National Institute of Standards and Technology, August 2004) ftp://sequoyah.nist.gov/pub/nist_internal_reports/ir_7151/ir_7151.pdf