The Ongoing Minutiae Interoperability Exchange Test (MINEX)

API Specification

Overview

The Minutiae Interoperability Exchange Test (MINEX) is an ongoing program to measure the performance of fingerprint matching software utilizing interoperable minutiae-based fingerprint templates. The content and format of those interoperable minutiae-based fingerprint templates are defined in this specification and are hereafter referred to as MINEX compliant templates.

Those wishing to submit software for MINEX testing shall be required to provide NIST with an SDK (Software Development Kit) library which complies with the API (Application Programmer Interface) specified in this document. At a minimum, the SDK submitted must provide functionality to create MINEX compliant templates based on individual fingerprint images. Support for matching pairs of MINEX compliant templates is encouraged, but optional.

In addition to providing a general platform for testing the performance of interoperable fingerprint systems, MINEX provides a mechanism for testing compliance with NIST Special Publication 800-76-1 [1] (refer to section 7.4).

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.

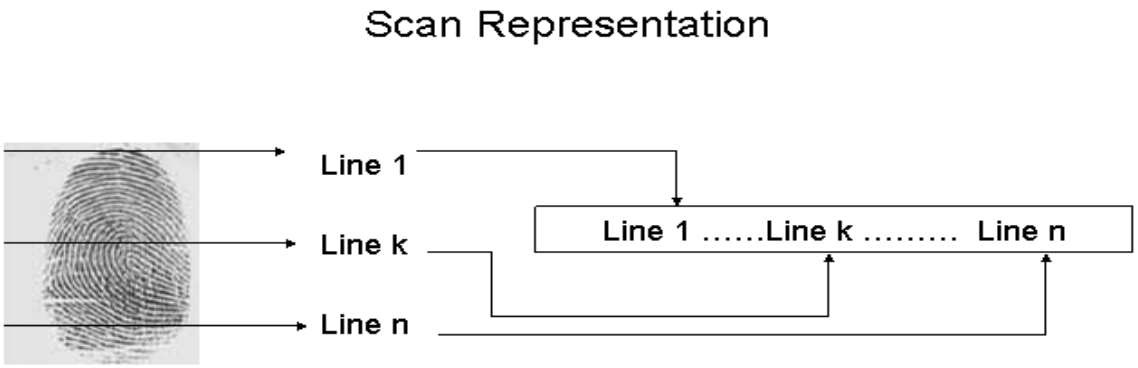

Figure 1 Order of scanned lines

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*

All images used for testing in MINEX come from the POEBVA data set described in NISTIR 7296 [2] (see Appendix B, Table 23 page 47) and thus have been obtained from live-scan sensors (Smiths-Heimann ACCO 1394 and Cross Match 300A). All images tested in MINEX are plain impression type images.

2 800-76-1 Compliant Templates

- To be considered MINEX compliant templates, all templates created must be compliant with NIST Special Publication 800-76-1 [1]. (refer to Table 11, page 22.)

Past participants in MINEX04 [2] may note that the requirements for templates specified by the Ongoing MINEX test are identical except for the fields listed below:

- In MINEX04, the field *Finger Quality* field had a range of values resulting from re-mapping the NIST NFIQ [3] quality values (1 through 5) to the values 100,75,50,25 and 1 respectively. However, 800-76-1 re-maps these same NFIQ quality values to 100,80,60,40, & 20 respectively.
- In MINEX04, the field *Impression Type* had a range of 0 through 3. However, 800-76-1 limits the range of values to 0 and 2.

3 Testing Interface Description

MINEX 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 MINEX testing interface:

```
// Finger quality values

#define QUAL_POOR       20       // NFIQ value 5
#define QUAL_FAIR       40       // NFIQ value 4
#define QUAL_GOOD       60       // NFIQ value 3
#define QUAL_VGOOD      80       // NFIQ value 2
#define QUAL_EXCELLENT  100      // NFIQ value 1

// Impression type codes

#define IMPTYPE_LP      0x00     // Live-scan plain
#define IMPTYPE_NP      0x02     // Nonlive-scan plain


// 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
```

3.2 Minutiae Extraction and Matching

3.2.2 Create Template

```
int32_t
create_template(const uint8_t* raw_image,
       const uint8_t   finger_quality,
       const uint8_t   finger_position,
       const uint8_t   impression_type,
       const uint16_t height,
       const uint16_t width,
       uint8_t *template);
```

Description

This function takes a raw image as input and outputs the corresponding MINEX compliant template. 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 will be used in later comparisons.

Note – *null templates* are defined as containing the Record header and Finger View header only, with zero minutiae information (i.e. *Number of Minutiae* shall be set to 0). Thus, it is a 32 byte template (26-byte Record Header + 4-byte Finger View header + 2 bytes for the Extended Data Block length which is 0x0000). All other fields in the Record and Finger View headers shall be set to their regular and accurate values.

Parameters

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

`finger_quality` **(input)**: The quality of the fingerprint image (e.g. `QUAL_GOOD`).

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

`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_t
match_templates(const uint8_t *probe_template,
        const uint8_t *gallery_template,
        float *score);
```

Description

This function compares two MINEX compliant templates and outputs a match score. The ***probe_template*** parameter shall be compared to the ***gallery_template*** parameter (in that precise order where the underlying matcher 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 (see 3.3 below).**

Parameters

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

`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 shall return a score of -1.

Note – If the legitimate range of match scores includes the value -1, the participant must inform the MINEX Test Liaison.

3.2.4 Get PIDs

```
int32_t
get_pids(uint32_t*  feature_extractor,
         uint32_t*  matcher);
```

Description

This function retrieves CBEFF PID information which identifies the SDK's core feature extractor and (if supported) template matcher. The PID output for ***feature_extractor*** shall be identical in both format and value to the CBEFF Product Identifier (PID) defined by INCITS 378-2004 [4] (refer to section 6.4.4). If the SDK supports template matching functionality the PID output for ***matcher*** shall have the two most significant bytes (specified by INCITS 378-2004 as identifying the "owner" of the technology) set to values identical to the corresponding bytes of ***feature_extractor***. Otherwise, if the SDK does not support matching functionality, the PID value returned for ***matcher*** shall be 0. . It may be assumed that memory for the ***feature_extractor*** and ***matcher*** parameters are allocated before the call. **Note that the two least significant bytes of the CBEFF PID are defined by INCITS 378-2004 as identifying the version of the feature extractor (referred to as PID "Type"). The two least significant bytes of *feature_extractor* and *matcher* shall be set as specified by INCITS 378-2004 (i.e. they may either be set to 0, or to a version number assigned by the "owner" of the technology).**

Parameters

`feature_extractor` **(output)**: A PID which identifies the SDK's feature extractor.

`matcher` **(output)**: A PID which identifies the SDK's matcher.

Return Value

This function returns *zero* on success or a documented *non-zero* error code on failure. In the latter case, both output parameters shall be set to 0.

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 code	Explanation
0	Success
1	Image size not supported
2	Failed to extract minutiae – unspecified error
3	Failed to extract minutiae – impression type not supported
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). 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 Cygwin1.7.x/GCC version 4.3.4 will be used; for CentOS 5.x/GCC 4.1.2 will be used*). For example,

```
g++  –o mintest  mintest.cpp  -L. –lminex
```

Participants are required to provide their library in a format that is linkable using g++ with the NIST test driver, which is compiled with g++. All compilation and testing will be performed on x86-64 platforms running either Windows Server 2003/2008 (running Cygwin 1.7.x under the WOW64 32-bit environment) or CentOS 5.x (dependent upon the operating system requirements of the SDK). Thus, participants are strongly advised to verify library-level compatibility with g++ (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.).

Note: The API symbols used within the submitted SDK should be "unmangled," meaning the compiler should produce a binary file with symbol names undecorated, as a traditional C compiler would. NIST's C++ test driver will include the prototypes for all API symbols within an extern "C" block.

4.2 Usage

The SDK must 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 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. This speed is based on the current processor being used for testing which varies as hardware is updated. Please contact the liaison for current hardware used..

References

[1] C. Wilson, et al., "Biometric Data Specification for Personal Identity Verification," NIST Special Publication 800-76-1

http://www.nist.gov/manuscript-publication-search.cfm?pub_id=51101

[2] P. Grother, et al., "Performance and Interoperability of the INCITS 378 Template," NIST IR 7296 http://www.nist.gov/manuscript-publication-search.cfm?pub_id=150619

[3] E. Tabassi, et al. "Finger Print Image Quality," NISTIR 7151 2004 (Gaithersburg, MD: National Institute of Standards and Technology, August 2004) http://www.nist.gov/manuscript-publication-search.cfm?pub_id=905710

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