IREX IX

Multi-Spectral Iris Evaluation
Concept, Evaluation Plan, and API Specification
Version 1.3

George W. Quinn, Patrick Grother, and James Matey

Image Group
Information Access Division
Information Technology Laboratory

December 13, 2016

Status of this Document

This is the first public version of this document. Comments and questions should be submitted to irex@nist.gov. This document can be downloaded from http://www.nist.gov/itl/iad/ig/irexix.cfm.

Timeline

Table 1: Milestones and deadlines

April 14th, 2016	NIST releases API version 0.1.
May 18th, 2016	Comments due on Initial API.
May 24th, 2016	Final API released.
October 7th, 2016	Submission deadline for Phase I.
November 21st, 2016	Target for NIST to provide participants with interim reports.
January 7th, 2017	Submission deadline for Phase II.
TBD	Target deadline for first public report.

Release Notes

NOTE: IREX IX is similar in many ways to IREX I and IV with respect to its API and implementation requirements.

Changes relating to version 1.3 of this document are highlighted with an orange background color. Notably:

1. *The deadline for submitting to Phase I has been extended a month.*
2. *In Section 3.4 the compiler has been corrected to g++ 2.8.5, the default version that comes with CentOS 7.2 (1511).*
3. *To clarify, a non-zero value for the* **failed** *attribute of the* **candidate** *structure means the candidate will be ignored (Section 5.2).*

Changes relating to version 1.2 of this document are highlighted with a blue background color. Notably:

1. *In Section 2.4.2 the test machines will be upgraded from CentOS 7.0 to CentOS 7.2. Any software developed for CentOS 7.0 should still work for CentOS 7.2.*
2. *In Section 3.6.2 submissions must remain operational until at least January 7th, 2018.*
3. *The Participation Agreement at the end of this document has been updated.*

Changes relating to version 1.1 of this document are highlighted with a pink background color. Notably:

1. *In Section 3.4 libraries should be linkable according to the C++11 linkage specification, not the C linkage specification.*

Changes relating to version 1.0 of this document are highlighted with a green background color. Notably:

1. *A more detailed description of the manually marked up iris boundary points is provided in Section 5.3: iris_boundary Struct Reference.*
2. *In the iris_sample stucture, a wavelength value of 0 now means the wavelength is either unspecified or unknown.*
3. *Participants are no longer required to submit Class A and Class B libraries at the same time (Section 3.1).*
4. *Section 3.6.4 now specifies that each submission (either Class A or Class B) shall be no more than a gigabyte.*
5. *Section **??** now specifies the pixel dimensions for the multi-spectral images.*
6. *Subsection 2.1.8: Ground Truth Integrity has been moved under Section 2.1: Performance Metrics.*
7. *The definitions of FNIR and FPIR have been clarified in Table 4. Both metrics are actually computed the same way as in IREX IV.*
8. *Additional text has been added to Subsection 2.1.8 explaining how NIST hopes to avoid having to resort to the horizontal flipping strategy used in IREX III and IV to mitigate the effects of ground truth errors.*

Changes relating to the second version of this document are highlighted with a yellow background color. Notably:

1. *NIST is targeting Nov 21st, 2016 to release interim reports.*
2. *Clarifications and corrections were made in sections 2.1.3: Single-eye and Dual-eye Testing and 2.1.1: Accuracy for One-to-one Matching.*
3. *Section 6 now specifies which machine will be used to test the speed requirements.*

Contents

Terms and Definitions

Table 2: The following terms and definitions are used in this document

ANSI	American National Standards Institute
API	Application Programming Interface
EDB	Enrollment Database
FNIR	False Negative Identification Rate
FPIR	False Positive Identification Rate
DET	Detection Error Trade-off
ISO	International Standards Organization
ISO/IEC 19794-6	ISO/IEC standard titled "Information technology - Biometric data interchange formats - Part 6: Iris image data"
ISO/IEC 29794-6	ISO/IEC standard titled "Biometric Sample Quality - Part 1: Framework"
IREX	Iris Exchange
NIST	National Institute of Standards and Technology

1 IREX IX Concepts

1.1 Overview

This document establishes a concept of operations (CONOPS) and application programming interface (API) for the Iris Exchange (IREX) IX Evaluation. IREX IX is a follow-up to IREX IV in the sense that it includes an analysis of one-to-many iris recognition for large-scale deployments. IREX IX also aims to test the performance of iris recognition over a multi-spectral iris dataset.

The goals of this evaluation are:

- To evaluate the current state of iris recognition with a focus on large-scale applications.
- To assess the degree to which iris recognition has advanced since IREX IV (~June 2012).
- To evaluate iris recognition over a "difficult" set of iris images that reflects common problems that occur during data collection (e.g. eyelid occlusion).
- To conduct a multi-spectral evaluation of iris recognition.

This marks the ninth installment of the IREX program. See http://www.nist.gov/itl/iad/ig/irexix.cfm for all IREX-related material.

1.2 Application Scenarios

Similar to IREX IV, part of the current evaluation will focus on practical applications of iris recognition with an emphasis on large-scale deployments (i.e. where the enrollment database contains up to several million subjects). The interest is in *one-to-many open-set* identification systems. Systems operating in a *one-to-many* mode (sometimes referred to as "identification mode") are tasked with identifying the individual without a prior claim to identity. *Open-set* means there is no guarantee that the searched individual is enrolled in the database.

IREX IX also includes a one-to-one matching component primarily intended to explore the ability of iris recognition to match samples collected over a range of wavelengths. Most iris cameras capture in the 700-850nm range of the electromagnetic spectrum. This evaluation will test how well recognition algorithms can segment and match iris samples captured over wavelengths ranging from 400nm to the infrared. Cross-spectral matching will also be tested. Additionally, the evaluation will test the ability of matchers to compare color-captured visible-spectrum images with possible applications in forensics iris.

1.3 The IREX Program

The IREX Program was initiated by NIST to support an expanded marketplace of iris-based applications. IREX provides quantitative support for iris recognition standardization, development, and deployment. To date, six activities have been completed and three more are tentatively planned. Each is summarized below.

- **IREX I** [1] was a large-scale, independently administered, evaluation of one-to-many iris recognition. It was conducted in cooperation with the iris recognition industry to develop and test standard formats for storing iris images. Standard formats are important for maintaining interoperability and preventing vendor lock-in. The evaluation was conducted in support of the ISO/IEC 19794-6 and ANSI/NIST-ITL 1-2011 standards.
- **IREX II** [2] supported industry by establishing a standard set of quality metrics for iris samples. Although iris recognition has the potential to be extremely accurate, it is highly dependent on the quality of the samples. The evaluation tested the efficacy of 14 automated quality assessment algorithms in support of the ISO/IEC 29794-6 standard [3].
- **IREX III** [4, 5] was a performance test of iris recognition algorithms over operational data. Despite growing interest in iris-based technology, at the time there was a paucity of experimental data to support published theoretical considerations and accuracy claims. IREX III constituted the first public presentation of large-scale performance results using operational data.
- **IREX IV** [6, 7] built upon IREX III as a performance test of one-to-many iris recognition. In addition to providing participants from previous evaluations an opportunity to further develop and test their recognition algorithms, this evaluation explored the potential for using a cost equation model for optimizing algorithms for specific applications.
- **IREX V** [8] provides best practice recommendations and guidelines for the proper collection and handling of iris images.
- **IREX VI** [9] is a temporal study of iris recognition. It assessed the degree to which accuracy decreases with the time lapsed between collection of initial enrollment and recognition images. The study aimed to quantify natural ageing effects in a healthy population only.
- **IREX VII** intends to define a framework for communication and interaction between components in an iris recognition system. By introducing layers of abstraction that isolate underlying vendor-specific implementation details, a system can become more flexible, extensible, and modifiable.

- **IREX VIII** will aim to support implementation of the iris recognition option extended under NIST Special Publication 800-76-2. This activity will constitute a laboratory evaluation of iris recognition algorithms capable of producing and consuming conformant instances of ISO/IEC 19794-6:2011.
- **IREX IX** is the current evaluation. Its novel component will be the testing of iris recognition over samples collected across a range of wavelengths (400nm and above). It will also test iris recognition over an operational set of iris samples, similar to IREX III and IV.

The latest information on the IREX Program can be found on the IREX website (http://www.nist.gov/itl/iad/ig/irexix.cfm).

2 Evaluation Overview

The evaluation will be conducted offline. Offline evaluations are attractive because they allow uniform, fair, repeatable, and convenient testing. However, they do not capture all aspects of an operational system. While this evaluation is designed to mimic operational reality as much as possible, it does not include a live image acquisition component or any interaction with real users.

2.1 Performance Metrics

2.1.1 Accuracy for One-to-one Matching

This evaluation will present core verification accuracy in the form of Detection Error Tradeoff (DET) curves [10], which show the tradeoff between the False Match Rate (FMR) and the False Non-match Rate (FNMR). If we define the null hypothesis as "the samples are from different people", then in the context of biometric matching, false matches are synonymous with Type I errors, and false non-matches are synonymous with Type II errors. DET curves are similar to Receiver Operation Characteristic (ROC) curves [11] but tend to more effectively highlight differences in the critical operating region. The IREX I final report [1] includes several examples of DET curves for iris recognition.

Table 3: DET accuracy metrics for one-to-one matching

Metric	Description
FMR	The fraction of non-mated comparisons that produce distance scores at or below threshold.
FNMR	The fraction of mated searches that produce distance scores above threshold.

2.1.2 Accuracy for One-to-many Matching

Accuracy will be measured for open-set applications, which means that no assumption can be made as to whether the searched individual is enrolled in the database. Most real-world applications of biometrics operate in this way (e.g. watchlists and de-duplication tasks). Closed-set applications, which assume that every searched individual is enrolled in the database (and thus only concern themselves with *which* of those enrollees the searched person matches best) are operationally uncommon and will not be tested.

Open-set biometrics systems are tasked with searching an individual against an enrollment database and returning zero or more candidates. Two types of decision errors are usually considered for this type of system. The first occurs when a candidate is returned for an individual that is not enrolled in the database. This is referred to as a

false positive. The second occurs when the correct candidate is not returned for an individual that is enrolled in the database. This is referred to as a false negative.

NIST will compute false positive statistics exclusively from non-mated searches and false negative statistics exclusively from mated searches. Although this reflects operational reality better than computing false positive statistics from mated searches (by simply ignoring correct mates on the candidate lists), it does not cover all factors that could affect the accuracy of a system (e.g. the position of the correct mate on the candidate list, the number of incorrect candidates returned for a mated search).

Core matching accuracy for identification-mode matching will be presented in the form of Detection Error Tradeoff (DET) curves [10] showing the tradeoff between the false positive identification rate (FPIR) and the false negative identification rate (FNIR). The Application Programming Interface (API) will require searches to return a fixed number of candidates but will only consider a candidate viable if its dissimilarity score is below some decision threshold. Table 4 defines how the accuracy metrics will be computed.

Table 4: DET accuracy metrics for one-to-many matching

Metric	Description
FPIR(t)	The fraction of non-mated searches for which at least one candidate has a distance score at or below threshold (t).
FNIR(t, r)	The fraction of mated searches for which the correct candidate is not on the list or has a distance score above threshold (t). The parameter r is the rank requirement and specifies the length of the candidate list. If the correct candidate is not on the list, then it fails the rank requirement. These definitions of FNIR and FPIR are the same as those in IREX IV.

In some plots, line segments will be drawn between curves to connect points of equal threshold. These line segments are intended to show how error rates at specific operating thresholds vary depending on factors such as the number of entries in the enrollment database or the quality of the iris samples.

2.1.3 Single-eye and Dual-eye Testing

NIST will evaluate performance for scenarios where:

- one iris sample is available per person.
- two samples (of opposite eyes) are available per person.

Due to the high frequency of erroneous (left/right) eye labelings in some of our test data, NIST will not provide labeling information for iris samples from this dataset. All samples will simply be labeled "unknown". NIST suspects the mislabelings are due to ambiguity with respect to whether "left" is intended to represent the subject's left eye (correct) or the eye on the left from the perspective of the camera operator (incorrect).

You may assume that for all our testing, whenever two-eyes are provided, they represent opposite eyes from the same person.

When testing single-eye performance, NIST will enroll left and right eyes of one person under different identifiers as though they came from different persons. This will allow NIST to test over larger enrollment databases. The test harness will never enroll two samples of the same iris under different identifiers.

2.1.4 Accuracy-speed Trade-off

NIST may perform an analysis of the trade-off between speed and accuracy as has been done in previous NIST evaluations.

2.1.5 Timing Statistics

NIST will report the computation time for all core functions of the implementations (e.g. feature extraction, searching). As was done in previous IREX evaluations, search time will be plotted as a function of enrollment size with a focus on whether the trend is sub-linear for any of the implementations. Batch mode processing, where more than one search is conducted at a time, will not be tested. Timing estimates will be made on an unloaded machine running a single process at a time. The machine's specifications are described in Section 2.4.1.

2.1.6 Template Sizes

The size of the proprietary templates generated by the implementations is relevant because it impacts storage requirements and computational efficiency. Therefore, NIST will report statistics on the size of enrollment and identification templates.

2.1.7 Runtime Memory Usage

NIST will monitor runtime memory usage during one-to-many searches and report the results.

2.1.8 Ground Truth Integrity

A hazard with collecting operational data is that ground truth identity labels can be incorrectly assigned due to clerical error. A Type I error occurs when a person's iris image is present under two or more identities. When possible, NIST will correct for this type of error by comparing samples from different datasets (where it is highly unlikely that any particular person is represented in both datasets). Type II errors occur when two or more persons are assigned the same subject identifier, which can lead to apparent false negatives. NIST cannot correct for this type of error.

2.2 Iris Datasets

At a minimum, NIST will test over samples having the following characteristics.

2.3 Multi-wavelength

Iris samples captured over a variety of wavelengths, from the visible to the infrared. Submissions should support arbitrary image pixel dimensions.

2.3.1 Operational Dataset

Operationally collected samples from cooperative users.

2.4 Test Environment

2.4.1 Hardware Specifications

NIST intends to support high performance by specifying the runtime hardware beforehand. There are several types of computer blades that may be used in the testing. The following list gives some details about the hardware of each blade type:

- Dell M610 - Dual Intel Xeon X5680 3.3 GHz CPUs (6 cores each).

- Dell M910 - Dual Intel Xeon X7560 2.3 GHz CPUs (8 cores each).
- Dual Intel Xeon E5-2695 3.3 GHz CPUs (14 cores each; 56 logical CPUs total).

Each CPU has 512K of cache. The bus runs at 667 Mhz. The main memory is 192 GB Memory as 24 8GB modules. We anticipate that 16 processes can be run without time slicing. NIST requires 64-bit implementations to support large memory allocation.

2.4.2 Operating System

The test machines will have CentOS 7.2 installed, which runs Linux kernel 3.10.0 (http://www.centos.org/). An ISO image of the distribution can be downloaded from NIGOS (http://nigos.nist.gov:8080/evaluations/CentOS-7-x86_64-Everything-1511.iso).

2.5 Reporting of Results

IREX IX will have two submission phases.

2.5.1 Final Report

Following completion of the testing, NIST will publish one or more Interagency Reports (IRs) on the results. NIST may also use the results to publish in other academic journals or present at conferences or workshops.

2.5.2 Interim Reports

NIST will provide participants with "score-card" performance results for Phase I submissions. These interim reports will be sent as they become available, so participants who submit earlier are more likely to receive their results sooner. A participant may submit up to two Class A and two class B libraries (during Phase I) followed by up to two more Class A and B libraries after receipt of the interim report (during Phase II). To receive an interim report, Phase I libraries must be submitted by October 7th, 2016.

While the score cards can be used by the participants for arbitrary purposes, they are intended to promote development and to provide the participants with a faster turnaround on how well their implementations performed. Score cards will be auto-generated for each implementation and will 1) include timing, accuracy, and other performance statistics, 2) include results from other participants without identifying them, 3) be expanded and modified as additional analyses are performed, and 4) be released asynchronously with implementation submissions. NIST does not intend to release the score cards publicly, though it may show them to U.S. government test sponsors. While the score cards are not intended for wider distribution, NIST can only request that sponsoring agencies not release their content.

3 Software Submission

3.1 Participation Requirements

Participation is open to any commercial organization or academic institution that has the ability to implement a large-scale one-to-many iris identification algorithm. There is no charge and participation is open worldwide.

The following rules apply:

- Participants must complete and submit the Participation Agreement.

- Participants must submit at least one Class A and one Class B implementation during Phase I or Phase II. Class A and B libraries ***need not*** be submitted concurrently.
- Participants are permitted to submit up to two class Class A and two Class B implementations for each phase (so up to eight submissions in total are permitted).
- Participants must adhere to the cryptographic protection procedures when submitting their implementations (see Section 3.2).
- All implementations must successfully validate to ensure their proper operation.

3.2 Submission Procedure

All software, data, and configuration files submitted to NIST must be signed and encrypted. Signing is performed to ensure authenticity of the submission (i.e. that it actually belongs to the participant). Encryption is performed to ensure privacy. The full process is described at http://biometrics.nist.gov/cs_links/iris/irex/NIST_biometrics_crypto2.pdf.

Note: NIST will not accept any submissions that are not signed and encrypted. NIST accepts no responsibility for anything that occurs as a result of receiving files that are not encrypted with the NIST public key.

Implementations shall be submitted to NIST as encrypted *gpg* files. If the encrypted implementation is below 20MB, it can be emailed directly to NIST at irex@nist.gov. If the encrypted implementation is above 20MB, it can either be provided to NIST as a download from a webserver[1], or mailed as a CD/DVD to the following address:

```
IREX IX Test Liason (A214)
100 Bureau Drive
A214/Tech225/Stop 8940
NIST
Gaithersburg, MD 20899-8940
USA
```

Upon receipt, NIST will validate the implementation to ensure its correct operation. The validation process involves running the implementation over a small sample of test data. This test data will be provided to the participant, who must run the implementation in-house and provide NIST with the comparison results. NIST will then verify that the participant's in-house results are consistent with the output produced on the NIST blades. The test data along with full instructions will be posted on the IREX IX homepage (http://www.nist.gov/itl/iad/ig/irexix.cfm) as part of a validation suite.

3.3 Requirements for Library Submissions

Participants shall provide NIST will pre-compiled and linkable libraries. Dynamic libraries are permitted, but static ones are preferred. Participants shall *not* provide any source code. Header files should not be necessary, but if provided, should not contain intellectual property of the company nor any material that is otherwise proprietary.

NIST is testing both one-to-one (Class A) and one-to-many (Class B) submissions. Participants must submit at least one Class A library that adheres to the API specification in section 4.3. All libraries shall adhere to the naming convention described in Table 5. Additional dynamic or shared library files may be submitted that support this core library.

Implementation libraries must be 64-bit. This will support large memory allocations that are necessary when an enrollment database contains millions of entries. To achieve faster running times, NIST expects implementations will load the enrollment templates into main memory before the enrollment database is searched. It is safe to

[1] NIST shall not be required to register or enroll in any kind of membership before downloading the implementation.

Table 5: Naming convention for an implementation library.

Form:	libIREX_provider_class_sequence.suffix				
Part:	libIREX	provider	class	sequence	suffix
Description:	First part of the name, fixed for all submissions	a single word name of the main provider. *EXAMPLE:* thebes	The library class, 'A' for one-to-one (verification mode), and 'B' for one-to-many (identification mode).	A two-digit decimal identifier starting at 00 and incrementing any time a new submission is sent to NIST	Either .so or .a
Example:	libIREX_thebes_A_01.a				

assume that NIST will not build enrollment databases containing more than 10 million entries (generated from 10 million iris samples). This means that template sizes should not exceed ~19K on average.

NIST will ignore requests to alter parameters by hand (e.g. modify specific lines in an XML configuration file). Any such adjustments must be submitted as a new implementation.

3.4 Linking Requirements

NIST will link the submitted library file(s) to our ISO 2011 C++ language test drivers. Participants are required to provide their libraries in a format that is linkable using g++ version 4.8.5 .

Participants may provide customized command-line linking parameters. A typical link line might be:

```
g++ -I. -Wall -m64 -o irex_main irex_main.c -L. -lirex_thebes_A_01 -lpthread
```

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 (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.). Intel ICC is not available. Access to GPUs is not permitted. Intel Integrated Performance Primitives (IPP) libraries are permitted if they are delivered as part of the developer-supplied library package. It is the provider's responsibility to establish proper licensing of all libraries.

Libraries must export their functions according to the C++11 linkage specified in the API.

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 after receiving prior approval from the test liaison. Image processing libraries such as libpng and NetPbm should not be required since NIST will handle image reading and decompression.

IMPORTANT: Windows machines will not be used for testing. Windows-compiled libraries are not permitted. All software must run under LINUX.

3.5 Single-thread Requirement

Implementations must run in single-threaded mode.

3.6 Installation Requirements

3.6.1 Installation Must be Simple

Installation shall require the simple copying of files followed by a linking operation. There shall be no need for interaction with the participant provided everything goes smoothly. It shall not require an installation program.

3.6.2 No License Requirements or Usage Restrictions

The implementation shall allow itself to be executed on any number of machines without the need for machine-specific license control procedures or activation. The implementation shall neither implement nor enforce any usage controls or restrictions based on licenses, number of executions, presence of temporary files, etc. No activation dongles or other hardware shall be required. The implementations shall remain operable until at least January 7th, 2018 .

3.6.3 Sufficient Documentation Must be Provided

Documentation should be provided for all (non-zero) participant-defined error or warning return codes.

3.6.4 Disk-Space Limitations

The implementation may use configuration files and supporting data files. The total size of all libraries and configuration and data files (for one Class A or Class B submission) shall be no more than a gigabyte.

3.7 Runtime Behavior Requirements

NOTE: If an implementation is buggy or does not comply with these requirements, NIST may not test or report results for the implementation in publications.

3.7.1 No writing to Standard Error or Standard Output

The implementation will be tested in a non-interactive "batch" mode without terminal support. Thus, the submitted library shall run quietly (i.e. it should not write messages to "standard error" or "standard output". An implementation may write debugging messages to a log file. This log file must be declared in the documentation.

3.7.2 Exception Handling Should be Supported

The implementation should support error/exception handling so that, in the case of an unexpected error, a return code is still provided to the calling application. The NIST test harness will gracefully terminate itself if it receives an unexpected return code, as it usually indicates improper operation of the implementation.

3.7.3 No External Communication

Implementations running on NIST hosts shall not side-effect the runtime environment in any manner except through the allocation and release of memory. Implementations shall not write any data to an external resource (e.g. a server, connection, or other process). Implementations shall not attempt to read any resource other than those explicitely allowed in this document. If detected, NIST reserves the right to cease evaluation of the software, notify the participant, and document the activity in published reports.

3.7.4 Components Must be Stateless

All implementation components shall be "stateless" except as noted elsewhere in this document. This applies to iris detection, feature extraction and matching. Thus, all functions should give identical output, for a given input, independent of the runtime history. NIST will institute appropriate tests to detect stateful behavior. If detected, NIST reserves the right to cease evaluation of the software, notify the participant, and document the activity in published reports.

3.7.5 No Switches or Command-line Options

Each implementation must be capable of running stand-alone (i.e. no two submissions shall depend on the same copies of libraries or configuration files). Each implementation shall support only one "mode" of operation. NIST will not entertain the option to "flip a switch" or modify a configuration file to produce a new implementation. Rather, the participant must submit each "mode" as a separate implementation.

3.7.6 Handling Large Enrollment Templates

Enrollment templates should not require more than 200K of persistent storage, on average, per enrolled image. Participants should inform NIST if their implementations require more than 100K of persistent storage.

3.7.7 Minimum Speed Requirements

The implementations shall perform operations within the time constraints specified by Table 6. These time limits apply to the function call invocations defined in Section 6 using a Dell M910 system described in Section 2.4.1. Since NIST cannot regulate the maximum runtime per operation, limitations are specified as 90th percentiles (i.e. 90% of all calls to the function shall complete in less time than the specified duration). The limitations assume each template was generated from a single iris image.

Table 6: Time limitations for specific operations.

Library Class	Operation	Timing Restriction
Both	Creation of an enrollment template from a single 640x480 pixel image	1,000 ms
Both	Creation of an identification template from a single 640x480 pixel image	1,000 ms
Class A	Comparison between two templates generated from single image each	20 ms
Class B	Finalization of a 1 million template enrollment database	7,200,000 ms
Class B	Search duration on a database of one million templates	20,000 ms

3.7.8 Failed Template Generations

When the implementation fails to produce an enrollment template, it shall still return a blank template (which can be zero bytes in length). For the one-to-many library, the template will be included in the manifest like all other enrollment templates, but is not expected to contain any feature information.

4 API Specification

4.1 Overview

The design of this API reflects the following testing objectives:

- Support distributed enrollment on multiple machines, with multiple processes running in parallel.
- Support graceful failure recovery and the ability to log the frequency of errors.
- Respect the black-box nature of proprietary templates.
- Provide flexibility and freedom to the participant to use arbitrary algorithms.
- Support the ability to collect timing statistics for specific operations.
- Support the ability to collect statistics on template sizes.

4.2 Class A (one-to-one) Functions

Class A library submissions must export and properly implement all of the functions defined in this subsection. The testing process will proceed in two phases: (1) feature extraction and template generation followed by (2) template comparison. The order in which the test harness will call the functions is outlined in Table 7.

Table 7: Program Flow

Stage	Function	Metrics of Interest
Feature Extraction	**convert_multiiris_to_enrollment_template** Generates an enrollment template from one or more images of an individual. The implementation must be able to handle multiple calls to this function from multiple instances of the calling application.	Statistics on template size and generation time.
	convert_multiiris_to_verification_template Generates a verification template from one or more images of an individual.	Statistics on template size and generation time.
Comparison	**match_templates** Compares a verification template to an enrollment template.	Statistics on comparison time and accuracy.

Functions

- int32_t get_pid (std::string &sdk_identifier, std::string &email_address)

  Retrieves a self-assigned identifier and contact email address for the software under test.
- int32_t get_max_template_sizes (uint32_t &max_enrollment_template_size, uint32_t &max_verification_template_size)

  Retrieves the maximum (per-image) enrollment and search template sizes.
- int32_t convert_multiiris_to_enrollment_template (const std::vector< iris_sample > &input_irides, uint32_t &template_size, uint8_t *enrollment_template)

  Generates an enrollment template from a vector of iris samples.
- int32_t convert_multiiris_to_verification_template (const std::vector< iris_sample > &input_irides, uint32_t &template_size, uint8_t *verification_template)

  Generates a verification template from a vector of iris samples.
- int32_t match_templates (const uint8_t *verification_template, const uint32_t verification_template_size, const uint8_t *enrollment_template, const uint32_t enrollment_template_size, double &dissimilarity)

  Searches a template against an enrollment template and produces a dissimilarity score.

4.2.1 Detailed Description

4.2.2 Function Documentation

4.2.2.1 int32_t IREX9::get_pid (std::string & *sdk_identifier,* std::string & *email_address*)

Retrieves a self-assigned identifier and contact email address for the software under test.

Parameters

out	*sdk_identifier*	A hexidecimal integer stored as a null terminated ASCII string (e.g. "01\0"). The value can be whatever the participant chooses, but must be unique for each implementation.
out	*email_address*	The point of contact for the software under test, stored as a null terminated ASCII string.

Returns

Zero indicates success. Other values indicate a vendor-defined failure.

4.2.2.2 int32_t IREX9::get_max_template_sizes (uint32_t & *max_enrollment_template_size,* uint32_t & *max_verification_template_size*)

Retrieves the maximum (per-image) enrollment and search template sizes.

These values will be used by the test harness to pre-allocate space for template data. For a vector of K iris samples, the test-harness will pre-allocate K times the provided value before calling convert_multiiris_to_enrollment_template() or convert_multiiris_to_identification_template().

Parameters

out	*max_enrollment_template_size*	The maximum (per-image) size of an enrollment template in bytes.
out	*max_verification_template_size*	The maximum (per-image) size of a search template in bytes.

Returns

Zero indicates success. Other values indicate a vendor-defined failure.

4.2.2.3 int32_t IREX9::convert_multiiris_to_enrollment_template (const std::vector< iris_sample > & *input_irides,* uint32_t & *template_size,* uint8_t * *enrollment_template*)

Generates an enrollment template from a vector of iris samples.

If the function returns a zero exit status, the template will be used for matching. If the function returns a value of 8, NIST will debug. Otherwise, a non-zero return value will indicate a failure to acquire and the template will *not* be used in subsequent search operations.

Parameters

in	*input_irides*	The iris samples from which to generate the template.
out	*template_size*	The size, in bytes, of the output template.
out	*enrollment_template*	Template generated from the iris samples. The template's format is proprietary and NIST will not access any part of it other than to store it in the EDB. The memory for the template will be pre-allocated by the NIST test harness. The implementation shall *not* allocate this memory.

Returns

Return Value	Meaning
0	Success.
2	Elective refusal to process the MULTIIRIS.
4	Involuntary failure to extract features.
6	Elective refusal to produce a non-blank template.
8	Cannot parse the input data.
Other	Vendor-defined failure.

4.2.2.4 int32_t IREX9::convert_multiiris_to_verification_template (const std::vector< iris_sample > & *input_irides,* uint32_t & *template_size,* uint8_t * *verification_template*)

Generates a verification template from a vector of iris samples.

If the function returns a zero exit status, the template will be used for matching. If the function returns a value of 8, NIST will debug. Otherwise, a non-zero return value will indicate a failure to acquire and the template will *not* be used in subsequent search operations.

Parameters

in	*input_irides*	The iris samples from which to generate the template.
out	*template_size*	The size, in bytes, of the output template
out	*verification_template*	Template generated from the iris samples. The template's format is proprietary and NIST will not access any part of it other to pass it to identify_template() and possibly store it temporarily. The memory for the template will be pre-allocated by the NIST test harness. The implementation shall *not* allocate this memory.

Returns

Return Value	Meaning
0	Success.
2	Elective refusal to process the MULTIIRIS.
4	Involuntary failure to extract features.
6	Elective refusal to produce a non-blank template.
8	Cannot parse the input data.
Other	Vendor-defined failure.

If there are multiple iris samples, then a zero status should be returned as long as feature information could be

extracted from at least one of the images.

4.2.2.5 int32_t IREX9::match_templates (const uint8_t * *verification_template,* const uint32_t *verification_template_size,* const uint8_t * *enrollment_template,* const uint32_t *enrollment_template_size,* double & *dissimilarity*)

Searches a template against an enrollment template and produces a dissimilarity score.

Parameters

in	*verification_template*	A template generated by a call to convert_multiiris_to_verification_template().
in	*verification_template_size*	The size, in bytes, of the verification template.
in	*enrollment_template*	A template generated by a call to convert_multiiris_to_enrollment_template().
in	*enrollment_template_size*	The size, in bytes, of the enrollment template.
out	*dissimilarity*	A non-negative measure of the amount of dissimilarity between the templates.

Returns

Return Value	Meaning
0	Success.
2	One or more of the input templates were the result of a failed feature extraction.
Other	Vendor-defined failure.

4.3 Class B (one-to-many) Functions

Class B library submissions must export and properly implement all of the functions defined in this subsection. The testing process will proceed in two phases: (1) feature extraction and template generation followed by (2) template comparison. The order in which the test harness will call the functions is outlined in Table 13.

Table 13: Program Flow

Stage	Function	Metrics of Interest
Enrollment	**initialize_enrollment_session** Allows the implementation to perform initialization procedures. Provides the implementation with: • advanced notice of the number of individuals and images that will be enrolled. • read-only access to the participant-supplied configuration data directory. • read-only access to the directory where the enrollment database will reside.	

	convert_multiiris_to_enrollment_template Generates an enrollment template from one or more images of an individual. The implementation is permitted read-only access to the enrollment directory at this stage. The implementation must be able to handle multiple calls to this function from multiple instances of the calling application.	Statistics on template size and generation time.
	finalize_enrollment Constructs an enrollment database from the enrollment templates. Templates are provided to the function through a manifest file. The contents of the enrollment directory should be populated with everything that is necessary to perform searches against it. This function allows post-enrollment book-keeping, normalization, and other statistical processing of the templates.	
Pre-search	**initialize_feature_extraction_session** Prepares the implementation for the generation of identification templates. The implementation is allowed read-only access to the enrollment directory during this stage.	
	convert_multiiris_to_identification_template Generates an identification template from one or more images of an individual.	Statistics on template size and generation time.
Search	**initialize_identification_session** Prepares the implementation for searches against the enrollment database. The function may read data (e.g. templates) from the enrollment directory and load them into memory.	
	identify_template() Searches a template against the enrollment database and returns a list of candidates.	Statistics on search time and accuracy.

Functions

- int32_t get_pid (std::string &sdk_identifier, std::string &email_address)

  Retrieves a self-assigned identifier and contact email address for the software under test.
- int32_t get_max_template_sizes (uint32_t &max_enrollment_template_size, uint32_t &max_search_template_size)

  Retrieves the maximum (per-image) enrollment and search template sizes.
- int32_t initialize_enrollment_session (const std::string &configuration_location, const std::string &enrollment_directory, const uint32_t num_persons, const uint32_t num_images)

  Initialization function, called once prior to one or more calls to convert_multiiris_to_enrollment_template().
- int32_t convert_multiiris_to_enrollment_template (const std::vector< iris_sample > &input_irides, uint32_t &template_size, uint8_t *enrollment_template)

  Generates an enrollment template from a vector of iris samples.
- int32_t finalize_enrollment (const std::string &enrollment_directory, const std::string &edb_name, const std::string &edb_manifest_name)

Finalization function, used to construct an enrollment database from an EDB and its manifest.

- int32_t initialize_feature_extraction_session (const std::string &configuration_location, const std::string &enrollment_directory, uint64_t &expected_memsize)

  Initialization function, to be called once prior to one or more calls to convert_multiiris_to_identification_template().
- int32_t convert_multiiris_to_identification_template (const std::vector< iris_sample > &input_irides, uint32_t &template_size, uint8_t *identification_template)

  Generates an identification template from a vector of iris samples.
- int32_t initialize_identification_session (const std::string &configuration_location, const std::string &enrollment_directory)

  Initialization function, to be called once prior to one or more calls to identify_template().
- int32_t identify_template (const uint8_t *identification_template, const uint32_t identification_template_size, const uint32_t candidate_list_length, std::vector< candidate > &candidate_list)

  Searches a template against the enrollment database and returns a list of candidates.

4.3.1 Detailed Description

4.3.2 Function Documentation

4.3.2.1 int32_t IREX9::get_pid (std::string & *sdk_identifier,* std::string & *email_address*)

Retrieves a self-assigned identifier and contact email address for the software under test.

Parameters

out	*sdk_identifier*	A hexidecimal integer stored as a null terminated ASCII string (e.g. "01\0"). The value can be whatever the participant chooses, but must be unique for each implementation.
out	*email_address*	The point of contact for the software under test, stored as a null terminated ASCII string.

Returns

Zero indicates success. Other values indicate a vendor-defined failure.

4.3.2.2 int32_t IREX9::get_max_template_sizes (uint32_t & *max_enrollment_template_size,* uint32_t & *max_search_template_size*)

Retrieves the maximum (per-image) enrollment and search template sizes.

These values will be used by the test harness to pre-allocate space for template data. For a vector of K iris samples, the test-harness will pre-allocate K times the provided value before calling convert_multiiris_to_enrollment_template() or convert_multiiris_to_identification_template().

Parameters

out	*max_enrollment_template_size*	The maximum (per-image) size of an enrollment template in bytes.
out	*max_search_template_size*	The maximum (per-image) size of a search template in bytes.

Returns

Zero indicates success. Other values indicate a vendor-defined failure.

4.3.2.3 int32_t IREX9::initialize_enrollment_session (const std::string & *configuration_location,* const std::string & *enrollment_directory,* const uint32_t *num_persons,* const uint32_t *num_images*)

Initialization function, called once prior to one or more calls to convert_multiiris_to_enrollment_template().

The implementation shall tolerate execution of multiple calls to this function from different processes running on the same machine. Each process may be reading and writing to the enrollment directory.

Parameters

in	*configuration_location*	Path to a *read-only* directory containing vendor-supplied configuration parameters and/or runtime data files.
in	*enrollment_directory*	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. The software may populate this folder in any manner it sees fit.
in	*num_persons*	The number of persons who will be enrolled in the database.
in	*num_images*	The number of images, summed over all identities, that will be used to build the enrollment database.

Returns

Return Value	Meaning
0	Success
2	The configuration data is missing, unreadable, or in an unexpected format.
4	An operation on the enrollment directory failed (e.g. insufficient permissions, insufficient disk-space, etc).
6	The software cannot support the number of persons or images requested
Other	Vendor-defined failure

4.3.2.4 int32_t IREX9::convert_multiiris_to_enrollment_template (const std::vector< iris_sample > & *input_irides,* uint32_t & *template_size,* uint8_t * *enrollment_template*)

Generates an enrollment template from a vector of iris samples.

If the function returns a zero exit status, the calling application will store the template in the EDB, which is later be passed to finalize_enrollment(). If the function returns a value of 8, NIST will debug. Otherwise, a non-zero return value will indicate a failure to enroll. The template will still be added to the EDB and the manifest to ensure that an *N* person enrollment database contains *N* entries. If the function crashes, NIST will include a zero-length template in the EDB and the manifest. The finalization process must be able to process zero-length templates.

IMPORTANT: The implementation shall not attempt to write to the enrollment directory (nor to other resources) during this call. Data collected from the iris samples should be stored in the template or created from the templates during the finalization step.

Parameters

in	*input_irides*	The iris samples from which to generate the template.
out	*template_size*	The size, in bytes, of the output template.
out	*enrollment_template*	Template generated from the iris samples. The template's format is proprietary and NIST will not access any part of it other than to store it in the EDB. The memory for the template will be pre-allocated by the NIST test harness. The implementation shall *not* allocate this memory.

Returns

Return Value	Meaning
0	Success.
2	Elective refusal to process the MULTIIRIS.
4	Involuntary failure to extract features.
6	Elective refusal to produce a non-blank template.
8	Cannot parse the input data.
Other	Vendor-defined failure.

4.3.2.5 int32_t IREX9::finalize_enrollment (const std::string & *enrollment_directory,* const std::string & *edb_name,* const std::string & *edb_manifest_name*)

Finalization function, used to construct an enrollment database from an EDB and its manifest.

Finalization shall be performed after all enrollment processes are complete. It should populate the contents of the enrollment directory with everything that is necessary to perform searches against it. This function allows post-enrollment book-keeping, normalization, and other statistical processing of the generated templates. It should tolerate being called multiple times, altough subsequent calls should probably not do anything.

The format of the two input files is described in the table below. The enrollment database (EDB) file stores a concatenation of the templates generated by calls to convert_multiiris_to_enrollment_template() in binary format. It does not contain a header or any delimiters between templates. This file can potentially be several gigabytes in size. The EDB manifest is an ASCII file that stores information about each template in the EDB file. Each line contains three space-delimited fields specifying the id, length, and offset of the template in the EDB file. If the EDB file contains N templates, the manifest will contain N lines.

For all intents and purposes, the template id can be regarded as a person id.

Field	Description	Datatype Size
Template ID	Non-negative decimal integer, not necessarily zero-indexed or in any particular order.	4 bytes
Template Length	Non-negative decimal integer.	4 bytes
Offset of template in EDB file	Non-negative decimal integer.	8 bytes
Example: 901231 1024 0 5834891 0 1024 50403 1024 1024 ...		

Parameters

in	*enrollment_directory*	The top-level directory in which the enrollment database will reside. The implementation will have read and write access to this directory.
in	*edb_name*	The path to a single *read-only* file containing the concatenated templates. The implementation should extract content from this file and place it in the enrollment directory.
in	*edb_manifest_name*	The path to a single *read-only* file containing the EDB manifest.

Returns

Value	Meaning
0	Success.
2	Cannot locate the input data - the input files or names seem incorrect.
4	An operation on the enrollment directory failed.
6	One or more template files are in an incorrect format.
Other	Vendor-defined failure.

4.3.2.6 int32_t IREX9::initialize_feature_extraction_session (const std::string & *configuration_location,* const std::string & *enrollment_directory,* uint64_t & *expected_memsize*)

Initialization function, to be called once prior to one or more calls to convert_multiiris_to_identification_template().

The implementation shall tolerate execution of multiple calls to this function from different processes running on the same machine.

Parameters

in	*configuration_location*	Path to a *read-only* directory containing vendor-supplied configuration parameters and/or runtime data files.
in	*enrollment_directory*	The top-level directory in which the enrollment data was placed when finalize_enrollment() was called.
out	*expected_memsize*	Given the enrollment data, the implementation shall specify the expected or peak memory size (in bytes) that will be used during searching.

Returns

Return Value	Meaning
0	Success.
2	The configuration data is missing, unreadable, or in an unexpected format.
4	An operation on the enrollment directory failed.
Other	Vendor-defined failure.

4.3.2.7 int32_t IREX9::convert_multiiris_to_identification_template (const std::vector< iris_sample > & *input_irides,* uint32_t & *template_size,* uint8_t * *identification_template*)

Generates an identification template from a vector of iris samples.

If the function returns a zero exit status, the template will be used for matching. If the function returns a value of 8, NIST will debug. Otherwise, a non-zero return value will indicate a failure to acquire and the template will *not* be used in subsequent search operations.

Parameters

in	*input_irides*	The iris samples from which to generate the template.
out	*template_size*	The size, in bytes, of the output template
out	*identification_template*	Template generated from the iris samples. The template's format is proprietary and NIST will not access any part of it other to pass it to identify_template() and possibly store it temporarily. The memory for the template will be pre-allocated by the NIST test harness. The implementation shall *not* allocate this memory.

Returns

Return Value	Meaning
0	Success.
2	Elective refusal to process the MULTIIRIS.
4	Involuntary failure to extract features.
6	Elective refusal to produce a non-blank template.
8	Cannot parse the input data.
Other	Vendor-defined failure.

If there are multiple iris samples, then a zero status should be returned as long as feature information could be extracted from at least one of the images.

4.3.2.8 int32_t IREX9::initialize_identification_session (const std::string & *configuration_location,* const std::string & *enrollment_directory*)

Initialization function, to be called once prior to one or more calls to identify_template().

The function may read data (e.g. templates) from the enrollment directory and load them into memory.

Parameters

in	*configuration_location*	Path to a *read-only* directory containing vendor-supplied configuration parameters and/or runtime data files.
in	*enrollment_directory*	The top-level directory in which the enrollment data was placed when finalize_enrollment() was called.

Returns

Return Value	Meaning
0	Success.
Other	Vendor-defined failure.

4.3.2.9 int32_t IREX9::identify_template (const uint8_t * *identification_template,* const uint32_t *identification_template_size,* const uint32_t *candidate_list_length,* std::vector< candidate > & *candidate_list*)

Searches a template against the enrollment database and returns a list of candidates.

NIST will typically set the candidate list length to operationally feasible values (e.g. 20), but may decide to extend it to values that approach the size of the enrollment database.

Parameters

in	*identification_template*	A template generated by a call to convert_multiiris_to_identification_template().
in	*identification_template_size*	The size, in bytes, of the template.
in	*candidate_list_length*	The length of the candidate list array.
out	*candidate_list*	An array (of length candidate_list_length) of pointers to candidates. Each candidate shall be populated by the implementation and shall be sorted in ascending order of distance score (e.g. the most similar entry shall appear first). The candidate list must be populated with sensible values.

Returns

Return Value	Meaning
0	Success.
2	The input template is defective.
Other	Vendor-defined failure.

5 Supporting Data Structures

This section describes the data structures used by the API.

5.1 point Struct Reference

Defines a structure that specifies a coordinate in an image.

Public Attributes

- uint16_t x

  X-coordinate (0 == leftmost)
- uint16_t y

  Y-coordinate (0 == topmost)

5.1.1 Detailed Description

Defines a structure that specifies a coordinate in an image.

5.2 candidate Struct Reference

Defines a structure that holds a single candidate.

Public Attributes

- uint8_t failed

  Indicates whether the candidate is valid (0=valid, 1-255=invalid).
- uint32_t template_id

  Template identifier from the enrollment database.
- double distance_score

  Measure of distance between the searched template and the candidate.

5.2.1 Detailed Description

Defines a structure that holds a single candidate.

5.2.2 Member Data Documentation

5.2.2.1 uint8_t failed

Indicates whether the candidate is valid (0=valid, 1-255=invalid).

If this value is non-zero, the values for template_id and distance_score will be ignored.

5.2.2.2 uint32_t template_id

Template identifier from the enrollment database.

5.2.2.3 double distance_score

Measure of distance between the searched template and the candidate.

Lower scores indicate greater similarity. The distance score must be non-negative, unless the search template is somehow broken, in which case it shall be set to -1.

5.3 iris_boundary Struct Reference

Defines a structure that holds manual segmentation information for an iris sample.

Public Attributes

- point center

  Cordinate representing manual estimate of iris center.
- std::vector< point > pupil_boundary

  Vector of points outlining pupil boundary.
- std::vector< point > limbic_boundary

Vector of points outlining limbic (iris-sclera) boundary.

- std::vector< point > upper_eyelid_boundary

  Vector of points outlining upper eyelid boundary.

- std::vector< point > lower_eyelid_boundary

  Vector of points outlining lower eyelid boundary.

5.3.1 Detailed Description

Defines a structure that holds manual segmentation information for an iris sample.

If boundary information is unspecified or unknown, the center will be set to (-1,-1) and all vectors will be empty (i.e. have zero size).

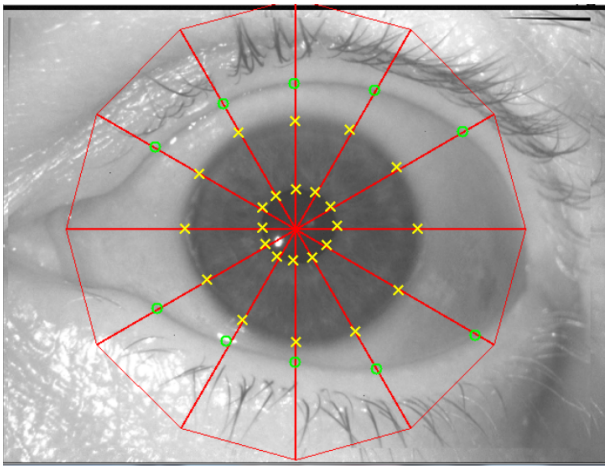

Figure 1: Depiction of how limbus, pupil-iris, and eyelid boundaries are manually identified in an iris sample. The yellow x's denote the limbus and pupil-iris boundaries and the green o's denote the eyelid boundaries. Although marked in the image, eyelid boundaries are only recorded where they overlap with the iris or pupil.

The center point is the inspecter's visual estimate of the center of the iris/pupil structure. The identification of this center is likely to be less robust than the manually identified boundary points.

The number of boundary points identified by the inspector varies from one sample to the next. The inspector identified up to 12 pupil-iris boundary points for a sample. When the eyelids occlude part of the pupil, they sometimes prevent some pupil-iris boundary points from being identified. The points are approximately equally spaced along the boundary.

On average the inspector identified about 12 points along the outer boundary. Outer boundary points mark either limbus or eyelid boundaries. Sometimes the inspector identified more than 12 points and sometimes fewer. Eyelid boundary points are only identified where they overlap the iris or pupil. On average, 8 limbus boundary points were identified per sample, 3 upper-eyelid boundary points where identified per sample, and 1 lower-eyelid boundary point was identified per sample.

We can make no guarantees with respect to the order of the boundary points in each vector. For example, we cannot guarantee they are ordered counter-clockwise around the center point. Although steps were taken during the manual markup proces to minimize the chances of making a mistake, we cannot guarantee to a certainty that all boundary points are properly located. NIST will fix mistakes if/when it finds them.

The validation package will contain a manual markup of a at least one iris sample for reference.

5.3.2 Member Data Documentation

5.3.2.1 point center

Cordinate representing manual estimate of iris center.

5.3.2.2 std::vector<point> pupil_boundary

Vector of points outlining pupil boundary.

5.3.2.3 std::vector<point> limbic_boundary

Vector of points outlining limbic (iris-sclera) boundary.

5.3.2.4 std::vector<point> upper_eyelid_boundary

Vector of points outlining upper eyelid boundary.

5.3.2.5 std::vector<point> lower_eyelid_boundary

Vector of points outlining lower eyelid boundary.

5.4 iris_sample Struct Reference

Defines a structure that holds a single iris with corresponding attributes.

Public Attributes

- eye_label eye
  The eye label for this iris sample (0 == undefined, 1 == right eye, 2 == left eye).
- uint16_t image_width
  Image width in pixels.
- uint16_t image_height
  Image height in pixels.
- uint16_t wavelength
  The wavelength in nanometers with which the iris sample was illuminated.
- uint8_t bit_depth
  Bit depth, 8 or 24 for RGB visible spectrum images.
- uint8_t * data
  Pointer to image raster data (in RGBRGBRGB...
- iris_boundary * **boundary**

5.4.1 Detailed Description

Defines a structure that holds a single iris with corresponding attributes.

5.4.2 Member Data Documentation

5.4.2.1 uint16_t image_width

Image width in pixels.

5.4.2.2 uint16_t image_height

Image height in pixels.

5.4.2.3 uint16_t wavelength

The wavelength in nanometers with which the iris sample was illuminated.

5.4.2.4 uint8_t bit_depth

Bit depth, 8 or 24 for RGB visible spectrum images.

5.4.2.5 uint8_t* data

Pointer to image raster data (in RGBRGBRGB...

format for 24-bit images).

6 References

[1] P. Grother, E. Tabassi, G. W. Quinn, and W. Salamon. Performance of Iris Recognition Algorithms on Standard Images. http://www.nist.gov/itl/iad/ig/irex.cfm, 2009. 2, 3

[2] E. Tabassi, P. Grother, and W. Salamon. IREX - IQCE performance of iris image quality assessment algorithms. Technical report, NIST, 2011. 2

[3] *ISO/IEC 29794-6 - Biometric Sample Quality Standard- Part 6: Iris Image*. Geneva, Switzerland, 2012. 2

[4] P. Grother, G.W. Quinn, J.R. Matey, M. Ngan, W. Salamon, G. Fiumara, and C. Watson. IREX: Performance of Iris Identification Algorithms. Technical report, NIST, 2011. 2

[5] G. Quinn and P. Grother. IREX III supplement I: Failure analysis. Technical report, NIST, 2011. 2

[6] P. Grother, G.W. Quinn, and M. Ngan. IREX: Evaluation of Iris Identification Algorithms. Technical report, NIST, 2013. 2

[7] P. Grother, G.W. Quinn, and M. Ngan. IREX: Compression Profiles for Iris Image Compression. Technical report, NIST, 2013. 2

[8] George W. Quinn, James Matey, Elham Tabassi, and Patrick Grother. IREX: Guidance for Iris Image Collection. Technical report, NIST, 2014. 2

[9] P. Grother, J. R. Matey, E. Tabassi, G. W. Quinn, and M. Chumakov. IREX: Temporal Stability of Iris Recognition Accuracy. Technical report, NIST, 2013. 2

[10] A. Martin, G. Doddington, T. Kamm, M. Ordowski, and M. Przybocki. The DET curve in assessment of detection task performance. In *Proc. Eurospeech*, pages 1895–1898, 1997. 3, 4

[11] J. A. Hanley and B. J. Mcneil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982. 3

Index

Application and Agreement to Participate in Iris Exchange IX

1. Who Should participate

1.1. Organizations ("Organizations") that develop iris matching software are eligible to participate.

1.2. Anonymous participation will not be permitted. This means that signatories to this document, *Application and Agreement to Participate in Iris Exchange IX*, acknowledge that they understand that the results (see Section 2.5) of the test of the Submission will be published with attribution to their Organization.

2. How to Participate

2.1. In order to participate in IREX IX, an Organization must provide the information requested in Section 9 of this Agreement (see below) identifying the Responsible Party, and the Point of Contact. Organization must also print and sign this Agreement, and send it to the location designated in Section 9.

2.1.1. The Responsible Party is an individual with the authority to commit the Organization to the terms in this Agreement.

2.1.2. The Point of Contact is an individual within the Organization with detailed knowledge of the participating Submission.

2.1.3. The Responsible Party and POC may be the same person.

2.2. Upon receipt of the signed Agreement by NIST, the Organization will be classified as a Participant in IREX IX. Applicants need only send one signed application; they do not need to send a new application for each submitted library.

2.3. Participant shall provide submissions ("Submissions") as specified in Section of the IREX IX Concept, Evaluation Plan, and API Specification ("Test Plan"). A Submission shall include all library files, configuration files, documentation, and all other files required by NIST and the Participant to validate and execute the tests specified in the Test Plan.

2.4. The Submission need not be used in a production system or be commercially available. However, the Submission must, at a minimum, be a stable implementation capable of conforming to the Test Plan that NIST has published for IREX IX.

2.5. The Submission must be encrypted before transmitting to NIST. Instructions for submitting can be found at http://biometrics.nist.gov/cs_links/iris/irex/NIST_biometrics_crypto2.pdf. Generic encryption instructions can be found in the Image Groups *Encrypting Software for Transmission to NIST* document available at http://www.nist.gov/itl/iad/ig/encrypt.cfm. A box for the Participants public key fingerprint is included on the Agreement. Submissions that are not signed with the public key fingerprint listed on the Agreement will not be accepted.

2.6. Submissions must be compliant with the Test Plan, NIST test hardware, and NIST test software.

3. Points of Contact

3.1. The IREX IX Liaison is the U.S. Government point of contact for IREX IX.

3.2. All questions should be directed to the irex@nist.gov, which will be received by the IREX IX Liaison and other IREX IX personnel.

4. Access to IREX IX Validation Samples

4.1. The IREX IX validation package is supplied to Participant to assist in preparing for IREX IX.

4.2. The iris samples in the IREX IX validation package are representative of the IREX IX test data only in format. Image quality, collection device, and other characteristics may vary between the validation samples and the test datasets.

5. Access to IREX IX Test Data

5.1. Participant will not have access to IREX IX Test Data.

6. Release of IREX IX Results

6.1. After the completion of IREX IX testing, the U.S. Government will publish all results obtained, along with the Organization's name on the IREX IX website.

6.2. Participant will be notified of the results via the Responsible Party and the Point of Contact provided on the Agreement.

6.3. After the release of the IREX IX results, Participant may use the results for their own purposes. Such results shall be accompanied by the following phtase: "Results shown from NIST do not constitute an endorsement of any particular system, produce, service, or company by the U.S. Government." Such results shall also be accompanied by the Internet address (URL) of the IREX IX website (http://www.nist.gov/itl/iad/ig/irexix.cfm).

7. Additional Information

7.1. Any data obtained during IREX IX, as well as any documentation required by the U.S. Government from the Participant (except the Submission), becomes the property of the U.S. Government. Participant will not acquire a proprietary interest in the data and/or submitted documentation. The data and documentation will be treated as sensitive information and only be used for the purposes of NIST tests.

7.2. Participant agrees that they will not file any IREX IX-related claim against IREX IX sponsors, supporters, staff, contractors, or agency of the U.S. Government, or otherwise seek compensation for any equipment, materials, supplies, information, travel, labor and/or other Participant-provided services.

7.3. The U.S. Government is not bound or obligated to follow any recommendations that may be submitted by the Participant. The U.S. Government, or any individual agency, is not bound, nor is it obligated, in any way to give any special consideration to Participant on future contracts.

7.4. By signing this Agreement, Participant acknowledge that they understand any test details and/or modifications that are provided in the IREX IX website supersede the information on this Agreement.

7.5. Participant may withdraw from IREX IX at any time before their Submission is received by NIST, without their participation and withdrawal being documented on the IREX IX website.

7.6. NIST will use the Participant's Submission only for NIST tests, and in the event errors are subsequently found, to re-run prior tests and resolve those errors.

7.7. NIST agrees not to use the Participant's Submission for the purposes other than indicated above, without the express permission by the Participant.

8. Reminders

8.1. NIST must receive the signed Agreement and Submission no later than *October 7th, 2016* for Phase I participation, and *January 7th, 2017* for Phase II participation (unless NIST extends these deadlines on the IREX IX webpage).

8.2. NIST requests that applicants send an email to irex@nist.gov after they have sent their applications. NIST will respond with a confirmation message upon receipt of the application.

8.3. See http://www.nist.gov/itl/iad/ig/irexix.cfm for the latest updates and information on IREX IX.

9. Application Submission

9.1. Please mail the completed and signed Agreement to:

```
IREX IX Test Liason (A214)
100 Bureau Drive
A214/Tech225/Stop 8940
NIST
Gaithersburg, MD 20899-8940
USA
```

Organization Name

Responsible Party

Full Name		
Address (Line 1)		
Address (Line 2)		
Address (Line 3)		
Phone Number	Fax Number	E-mail Address

Point of Contact Check if same as Responsible Party above: ☐

Full Name		
Address (Line 1)		
Address (Line 2)		
Address (Line 3)		
Phone Number	Fax Number	E-mail Address

Participant **must** complete the box below per the instructions for transmission of encrypted content to NIST, as defined at http://biometrics.nist.gov/cs_links/iris/irex/NIST_biometrics_crypto2.pdf. If preferred, Participant may fax their public key fingerprint to the IREX IX Liaison at (301) 975-5287.

Public Key Fingerprint

Participant
NIST A75C EECD EF65 3197 7E66 A960 67D0 4015 407A D929

With my signature, I hereby request consideration as a Participant in the Iris Exchange (IREX) IX Evaluation, and I am authorizing my Organization to participate in IREX IX according to the rules and limitations listed in this document.

With my signature, I also state that I have the authority to accept the terms stated in this Agreement.

Signature of Responsible Party Date