IREX IV

Evaluation of One-to-Many Iris Recognition
Concept, Evaluation Plan, and API Specification
Version 0.1

George W. Quinn and Patrick Grother

Image Group
Information Access Division
Information Technology Laboratory

National Institute of Standards and Technology
U.S. Department of Commerce

April 15th, 2012

Status of this Document

This is the first public version of this document. Comments and questions should be submitted to irex@nist.gov. The document can be downloaded from http://iris.nist.gov/irex.

Timeline

Table 1: Milestones and deadlines

April 16, 2012	NIST releases API version 0.1
May 1, 2012	Comments due on Initial API
May 15 - July 15, 2012	Anticipated submission window.

Release Notes

NOTE: IREX IV is very similar to IREX III with respect to its API and implementation requirements. Notable changes are highlighted throughout this document with a yellow background color.

Contents

Terms and Definitions

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

ANSI	American National Standards Institute
ANSI/NIST Type 17	American National Standard for Information Systems - Data Format for the Interchange of Fingerprint, Facial, and Other Biometric Information - Part 1
API	Application Programming Interface
EDB	Enrollment Database
FNIR	False Negative Identification Rate
FPIR	False Positive Identification Rate
FTS	Failure to search
FTX	Failure to extract features from an enrollment image
DET	Detection Error Tradeoff
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 Institution of Standards and Technology
UID	India's Unique Identity scheme

1 IREX IV Concepts

1.1 Overview

This document establishes a concept of operations (CONOPS) and application programming interface (API) for the Iris Exchange (IREX) IV Evaluation. IREX IV will be a large-scale evaluation of iris recognition technology over operational data. Like IREX III [1], it will focus exclusively on one-to-many applications.

The goals of this evaluation are

- To investigate the use of cost parameters for application specific optimization (see Section 2.1.2).
- To establish a compression profile for the efficient and compact storage of iris images (see Section 2.1.3).
- To measure the speed and accuracy of iris matchers over the OPS-II dataset of operational iris images.

This marks the fourth installment in the IREX program (see Figure 1). See http://iris.nist.gov/irex for all IREX related documentation.

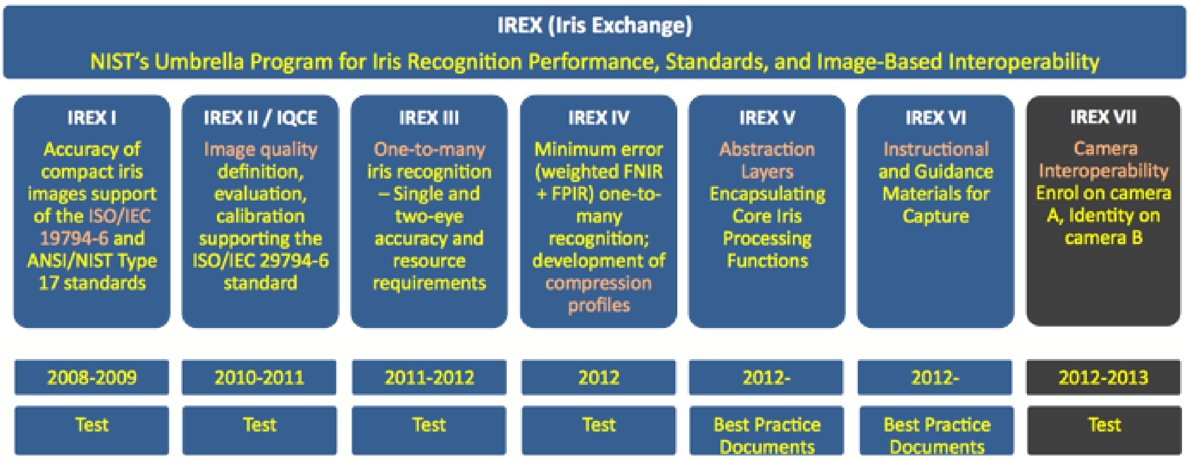
1.2 Market Drivers

This evaluation is intended to support a plural marketplace of iris recognition systems. While the largest applications, in terms of revenue, have been for border control and war zone identity management, India's Unique Identity (UID) scheme is currently using iris (in conjunction with fingerprints) for de-duplication on a massive scale.

The expanding marketplace for iris recognition has fueled the development of iris cameras designed to operate in a variety of applications. For example:

- Some standoff-capture cameras can rapidly image and verify (in a one-to-many mode) high volumes of people.

Figure 1: Current extent of the IREX program as well as planned expansions.



- Some mobile cameras can be preloaded with firm-ware based segmentation and identification capability for rapid one-to-many watchlist searches.

These applications are differentiated by population size, hardware capabilities, quality of the iris samples, and other variables.

1.3 Application Scenarios

The 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. To explore the potential for application-specific algorithm optimization, participants will submit two classes of implementations, each focusing greater attention on reducing a different type of error (see section 2.1.2). Table 3 details the parameters of this evaluation.

Participants may also submit implementations that perform cropping and masking of the iris images to convert them into an ISO/IEC 19794-6 compact format. Representing iris images compactly is crucial for applications operating over limited-bandwidth networks. India's Unique Identity (UID) scheme is seeking to reduce bandwidth requirements for the transmission of iris data.

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.

Table 3: Application Parameters

Parameter	Class P (Positive Identification System)	Class N (Negative Identification System)
Application Type	One-to-many open-set identification systems (e.g. watchlists, de-duplication operations).	
Class Description	High cost associated with false positives	High cost associated with false negatives
Example Applications	Biometric authentication for restricted access to high value information, resources, or facilities.	Watchlists for high-profile individuals. Investigational-mode searches.
Enrolled Database Size	Anywhere from $O(10^2)$ to $O(10^7)$ subjects.	
Prior NIST References	IREX III Final Report [1] IREX III Supplement I: Failure Analysis [2] Multiple Biometric Evaluation (MBE) 2010 [3]	
Performance Criteria	Primarily accuracy and speed. Also, memory usage, scalability, template-size, etc.	

2.1 Performance Metrics

2.1.1 Accuracy

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.

This evaluation will present core matching accuracy in the form of Detection Error Tradeoff (DET) [4] and Sensitivity-Reliability [5] plots, both of which show the tradeoff between the two types of error. 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 and SEL-REL accuracy metrics

Performance Plot	Metric	Description
Detection-error Tradeoff Curve	FPIR	The fraction of non-mated searches for which at least one candidate has a distance score at or below threshold.
	FNIR	The fraction of mated searches for which the correct candidate is not on the list or has a distance score above threshold.
Selectivity-Reliability Curve	SEL	The average number of candidates for a non-mated search having a distance score at or below threshold.
	REL	One minus FNIR

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.

2.1.2 Cost Function Optimization

This evaluation will investigate the use of cost parameters for application-specific algorithm optimization. The goal is to determine if matching algorithms can be modified to improve performance when the costs of errors are known in advance. The following cost model will be used as an evaluation metric for recognition performance:

$$E[\,\mathrm{Cost}(\tau)\,] = (1 - P_{Mated})\ \mathrm{FPIR}(\tau)\ C_P + P_{Mated}\ \mathrm{FNIR}(\tau)\ C_N \qquad (1)$$

where P_{Mated} is the *a priori* probability that the user is mated, C_P is the cost of a false positive, C_N is the cost of a false negative, $\mathrm{FPIR}(\tau)$ is the false positive identification rate, $\mathrm{FNIR}(\tau)$ is the false negative identification rate, and τ is the operating threshold. The model estimates the expected cost per user attempt, which could be a measure of time, workload, money, etc. The participant is tasked with minimizing the cost for a predetermined and fixed set of cost parameters (C_P, C_N, and P_{Mated}).

Cost parameters are often chosen to correspond to a specific application. Consider a biometric system that provides bank vault access to specific individuals. One might reasonably set the cost of a false positive to be the monetary value of whatever is in the vault, and the cost of a false negative to a value that reflects the amount of inconvenience incurred from having to open the vault by some other method. Setting P_{Mated} to 0.1 assumes that one out of every ten access attempts is by an allowed user.

NIST requires each participant to submit two implementations, each corresponding to a different set of cost parameters. These parameters are defined in Table 5. Class P implementations penalize false positives heavily and false negatives lightly. Class N implementations assign comparatively greater penalty to false negatives. For this class of implementations, suppression of false positives is less important. Both classes will be tested over one-eye and dual-eye tests. Participants may wish to use a different fusion rule for the two class types.

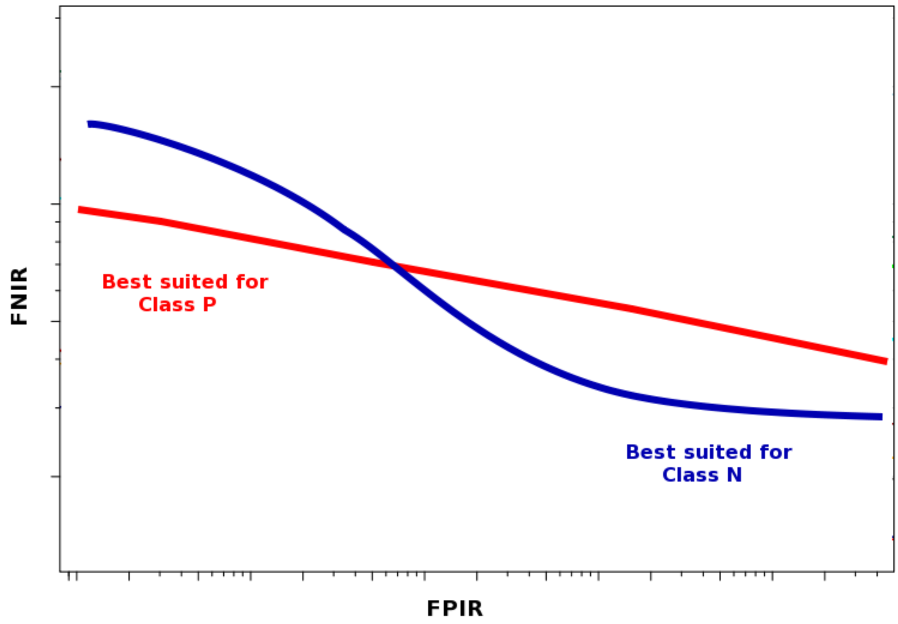
Table 5: Cost parameters for both submission types

Implementation Class	C_N	C_P	P_{mated}
Class P	1	10	0.01
Class N	200	1	0.1

Additionally, failures to extract (FTXs) and failures to search (FTSs) will be treated differently depending on the implementation class. For Class P implementations, both will be treated as failures in a *positive recognition system* (e.g. access control). This is the way NIST has handled FTXs and FTSs in prior evaluations. For Class N implementations, FTXs and FTSs be treated like failures in a *negative recognition system* (e.g. a watchlist). Failures in a negative recognition system increase the FPIR when they occur for non-mated searches, but do not increase the FNIR when they occur for mated searches. This differs from the way NIST has traditionally handled these types of failure.

The motivation for requiring participants to submit two implementations is to see if it is possible to change the shape of a DET to reduce cost for a specific set of cost parameters. Figure 2 plots standard DET curves for two identification algorithms. The two curves cross one another, making it impossible to state which is more accurate in any absolute sense. Since Class N implementations are penalized heavily for false negatives, and only lightly for false positives, both algorithms are expected to achieve their lowest cost toward the right end of the figure, where the blue curve performs better. Conversely, Class P implementations are penalized heavily for false positives but only lightly for false negatives. Thus, for this set of cost parameters, both algorithms are expected to achieve their lowest cost toward the left end of the figure, where the red curve performs better.

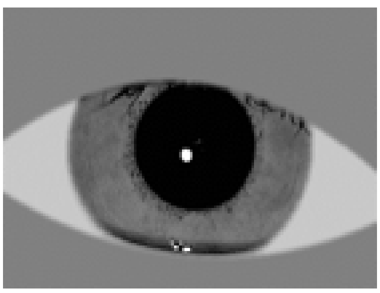
Figure 2: Notional DET plots demonstrating how the two classes place greater emphasis on different regions of the DET.



113 2.1.3 JPEG 2000 Compression

114 India's UID scheme will use the iris biometric for recognition tasks, and a desire has been expressed to represent iris feature information more compactly to reduce bandwidth usage during network transfer. The ideal solution is to store the images according to one of the compact and interoperable formats specified in ISO/IEC 19794-6. This evaluation seeks to further support the standard by establishing JPEG 2000 compression profiles for the efficient and compact storage of iris images. Toward this end, NIST will subject the images to lossy JPEG 2000 compression while tweaking various compression parameters. JPEG 2000 encoders that NIST may use include OpenJPEG [6] and Kakadu [7]. Participants are requested to submit implementations that can convert a raw iris image into an ISO/IEC 19794-6 Type 7 (cropped and masked) image (as shown in Figure 3). Support for this operation is optional but encouraged.

Figure 3: An example of an ISO/IEC 19794-6 Type 7 (cropped and masked) image.



115 2.1.4 Single-eye and Dual-eye Testing

116 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 the OPS-II dataset, NIST will no longer provide labeling information for iris samples. All samples will simply be labeled "U", indicating "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).

NIST will never provide more than two samples per person. Although eye labels will not be provided, it can be assumed that if two samples are provided, they represent opposite eyes of 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.

2.1.5 Accuracy-speed Trade-off

NIST will perform an analysis of the trade-off between speed and accuracy. However, participants are no longer requested to submit implementations of varying speeds. Rather, participants should submit different implementations that are each optimized to a different set of cost function parameters (see Section 2.1.2).

2.1.6 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 2.3.0.1.

2.1.7 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.

2.1.8 Runtime Memory Usage

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

2.1.9 Automated Quality Assessment

Automated quality assessment has a number of useful applications in iris recognition (e.g. determining in real-time whether a sample should be reacquired during a capture session). Automated quality assessment of iris samples was the primary focus of IREX II: IQCE [8]. In IREX IV, NIST will analyze the quality scores returned by the implementations during feature extraction. Error vs. reject curves, as described in [9], will be plotted. NIST may choose to perform additional analyses with an emphasis on how strongly quality scores correlate with matching accuracy. Support for automated quality assessment in the submitted implementations is optional.

2.2 Iris Datasets

2.2.1 The OPS-II Dataset

The primary test dataset for this evaluation is identical to the OPS dataset used in IREX III with one notable exception: The images in the current dataset were never compressed, while the vast majority of those in the original OPS dataset had been previously compressed using JPEG at a quality setting of 75.

The OPS-II consists of several million operational images collected from 18 distinct commercial iris cameras. Some subjects' irides were captured by more than one camera model. Most of the iris images have a pixel resolution of 640x480, but some are 480x480. NIST intends to exclude the pathological 330x330 images discussed in IREX III from this evaluation. Some of the non-pathological images still have poor sample quality (e.g. high amounts of occlusion, specular reflections, heavy pupillary constriction). Some were captured outside and contain heavily constricted pupils. See the IREX III Supplement I [2] for more information. Search and enrollment samples will be pulled from the same source and will therefore be of comparable quality.

2.2.2 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. To correct for this type of error during evaluation, NIST will estimate FPIR using search images that have been horizontally flipped[1]. The effect of flipping is discussed in the IREX III report. 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, but analyses in IREX III and its supplement indicate that Type II errors accounted for only a small fraction of the false negatives that occurred when the algorithms were tested over the OPS dataset.

2.3 Test Environment

2.3.0.1 Hardware Specifications

The test machines are high-end PC-class blades, each having 4 CPUs with 4 cores per CPU. The blades are labeled Dell M905, equipped with 4x Qual Core AMD Opteron 8376HE processors[2] running at 2.3GHz. Each CPU has 512K of cache. The bus runs at 667 MHz. Main memory consists of 192GB as 24 8GB modules. Sixteen processes can run without time slicing. NIST may use some test machines that have slightly different hardware specifications, but the operating system and compilation environment will remain homogenous across all blades. Furthermore, timing statistics will only be computed on machines having the aforementioned hardware specifications.

2.3.0.2 Operating System

The test machines will have CentOS 6.2 installed, which runs Linux kernel 2.6.32-220.7.1 (http://www.centos.org/).

2.4 Reporting of Results

2.4.1 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.

[1] Using the *jpegtran* application provided by the Independent JPEG Group, present on most LINUX platforms.

[2] `cat /proc/cpuinfo` returns `fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat npse36 clflush mmx fxsr sse sse2 ht syscall nx pni`

2.4.2 Interim Reports

NIST will provide participants with "score-card" performance results prior to the release of the final report. The interim reports will be sent as they become available, so participants who submit earlier are more likely to receive their results sooner. 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 form 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 (http://biometrics.nist.gov/cs_links/iris/irexIV/IREX_IV_Application_v1.pdf) .
- Participants must submit at least one Class P, and one Class N, implementation.
- Participants are permitted to submit up to two Class N and two Class P implementations (so up to four 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.

The deadline for submitting implementations will be posted to the IREX IV homepage (http://iris.nist.gov/irex/irexIV). NIST will not perform phased testing (i.e. the submission window will close before NIST provides participants with preliminary results).

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/irexIV/IREX_IV_Application_v1.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[3], or mailed as a CD/DVD to the following address:

```
IREX IV Test Liason (A214)
100 Bureau Drive
```

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

```
A203/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 IV homepage (http://iris.nist.gov/irexIV) 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.

At least one "core" library must be submitted that adheres to the API specification in section 4.2. This library shall adhere to the naming convention described in Table 6. Additional dynamic or shared library files may be submitted that support this core library.

Table 6: Naming convention for an implementation library.

Form:	libIREX_provider_class_sequence.suffix				
Part:	libIREX	provider	classes	sequence	suffix
Description:	First part of the name, fixed for all submissions	a single word name of the main provider. *EXAMPLE:* thebes	Functional class described in Table 5 (N or P).	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_N_03.a				

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 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 98/99 C/C++ language test drivers. Participants are required to provide their libraries in a format that is linkable using gcc version 4.1.2. The standard libraries are:

- `/usr/lib64/libstdc++.so.6.0.13 (GLIBCXX 3.4.13)`
- `/lib/libc.so.6 -> libc-2.12.so (GLIBC 2.12)`
- `/lib/libm.so.6 -> libm-2.12.so`

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

```
gcc -I. -Wall -m64 -o irex_main irex_main.c -L. -lirex_thebes_N_01 -lpthread
```

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 (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.). Intel IPP libraries are not permitted and will not be supplied. Intel ICC is not available. Access to GPUs is also not permitted.

On request, NIST will allow the use of g++ for linking, but the library must export its functions according to the C linkage specified by in the API. The Standard C++ library is available.

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. Multithreading was supported in IREX III, but all participants eventually chose to submit single-threaded implementations during the latter stages of phased testing.

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 October 31st, 2013.

3.6.3 Sufficient Documentation Must be Provided

Participants shall provide complete documentation of their implementations and detail any additional functionality or behavior beyond those specified here. The documentation must define all (non-zero) vendor-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 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 7. These time limits apply to the function call invocations defined in Section 7. 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 sample.

Table 7: Time limitations for specific operations.

Operaton	Timing Restriction
Creation of an enrollment template from a single 640x480 pixel image	1,000 ms
Creation of an identification template from a single 640x480 pixel image	1,000 ms
Finalization of a 1 million template enrollment database	7,200,000 ms
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). 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

Library submissions must export and properly implement all of the functions defined in this section. The testing process will proceed in two phases: (1) enrollment, followed by (2) identification. The order in which the test harness will call the functions is outlined in Table 8.

The design 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.

Table 8: 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.
Compression (optional)	**convert_raster_to_cropped_and_masked()** Converts a raw image to an ISO/IEC 19794-6 Type 7 (cropped and masked) image.	

4.2 Functions

Functions

- int32_t get_pid (char ∗sdk_identifier, char ∗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_recognition_template_size)

    Retrieves the maximum (per-image) enrollment and search template sizes.
- int32_t initialize_enrollment_session (const char ∗configuration_location, const char ∗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 MULTIIRIS ∗input_irides, MULTISEGMENTATION ∗output_properties, uint32_t ∗template_size, uint8_t ∗proprietary_template)

  Generates an enrollment template from a MULTIIRIS object.
- int32_t finalize_enrollment (const char ∗enrollment_directory, const char ∗edb_name, const char ∗edb_manifest_name)

  Finalization function, used to construct an enrollment database from an EDB and its manifest.
- int32_t initialize_feature_extraction_session (const char ∗configuration_location, const char ∗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 MULTIIRIS ∗input_irides, MULTISEGMENTATION ∗output_properties, uint32_t ∗template_size, uint8_t ∗identification_template)

  Generates an identification template from a MULTIIRIS object.
- int32_t initialize_identification_session (const char ∗configuration_location, const char ∗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, CANDIDATE ∗const ∗candidate_listm uint8_t &decision)

  Searches a template against the enrollment database and returns a list of candidates.
- int32_t convert_raster_to_cropped_and_masked (const ONEIRIS ∗input_iris, ONEIRIS ∗output_iris)

  Convert a raw (640x480 or 480x480) image to an ISO/IEC 19794-6 Type 7 (cropped and masked) image.

4.2.1 Function Documentation

4.2.1.1 int32_t get_pid (char ∗ *sdk_identifier,* char ∗ *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. The value can be whatever the participant chooses, but must be unique for each implementation. 5 bytes will be pre-allocated for this.
out	*email_address*	The point of contact for the software under test, stored as a null terminated ASCII string. 64 bytes will be pre-allocated for this.

Returns

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

4.2.1.2 int32_t get_max_template_sizes (uint32_t ∗ *max_enrollment_template_size,* uint32_t ∗ *max_recognition_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 MULTIIRIS containing K images, 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_-recognition_-template_size*	The maximum (per-image) size of a search template in bytes.

Returns

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

4.2.1.3 int32_t initialize_enrollment_session (const char ∗ *configuration_location,* const char ∗ *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

4.2.1.4 int32_t convert_multiiris_to_enrollment_template (const MULTIIRIS ∗ *input_irides,* MULTISEGMENTATION ∗ *output_properties,* uint32_t ∗ *template_size,* uint8_t ∗ *proprietary_template*)

Generates an enrollment template from a MULTIIRIS object.

In addition to handling raw OPS-II images, this function must be able to process ISO/IEC 19794-6 Type 7 (cropped and masked) images.

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 MULTIIRIS object 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`	*output_-properties*	Segmentation and quality information for each iris sample. The NIST test harness will pre-allocate the memory for the ONESEGMENTATION objects (one per ONEIRIS object). The calling application shall *NOT* initialize this memory.
`out`	*template_size*	The size, in bytes, of the output template.
`out`	*proprietary_-template*	Template generated from the MULTIIRIS object. 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 template.
8	Cannot parse the input data.
Other	Vendor-defined failure.

4.2.1.5 int32_t finalize_enrollment (const char $*$ *enrollment_directory,* const char $*$ *edb_name,* const char $*$ *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.

4.2.1.6 int32_t initialize_feature_extraction_session (const char * *configuration_location*, const char * *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.
`in`	*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.

4.2.1.7 int32_t convert_multiiris_to_identification_template (const MULTIIRIS ∗ *input_irides,* MULTISEGMENTATION ∗ *output_properties,* uint32_t ∗ *template_size,* uint8_t ∗ *identification_template*)

Generates an identification template from a MULTIIRIS object.

In addition to handling raw OPS-II images, this function must be able to process ISO/IEC 19794-6 Type 7 (cropped and masked) images.

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	*output_-properties*	Segmentation and quality information for each iris sample. The NIST test harness will pre-allocate the memory for the ONESEGMENTATION objects (one per ONEIRIS object). The implementation shall *NOT* initialize this memory.
out	*output_-properties*	Segmentation and quality information for each iris sample. The NIST test harness will pre-allocate the memory for the ONESEGMENTATION objects.
out	*template_size*	The size, in bytes, of the output template
out	*identification_-template*	Template generated from the MULTIIRIS object. 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 template.
8	Cannot parse the input data.
Other	Vendor-defined failure.

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

4.2.1.8 int32_t initialize_identification_session (const char ∗ *configuration_location,* const char ∗ *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.

4.2.1.9 int32_t identify_template (const uint8_t ∗ *identification_template,* const uint32_t *identification_template_size,* const uint32_t *candidate_list_length,* CANDIDATE ∗const ∗candidate_listm uint8_t & *decision*)

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. The memory for the candidates will be pre-allocated by the NIST test harness.
out	*decision*	A boolean decision on whether the implementation believes the top ranked candidate matches the identification template (1=yes, 0=no). This decision should attempt to minimize the cost function for the given class type (see Section 2.1.2).

Returns

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

457 4.2.1.10 int32_t convert_raster_to_cropped_and_masked (const ONEIRIS ∗ *input_iris*, 458 ONEIRIS ∗ *output_iris*)

459 Convert a raw (640x480 or 480x480) image to an ISO/IEC 19794-6 Type 7 (cropped and masked) image.

460 This function shall perform the same operations that were required to generate a KIND 7 record in IREX I. This involves cropping the image and masking the sclera and eyelids with a solid color. As described in ISO/IEC 19794-6, cropping shall provide a margin 0.6R wide on both the left and right sides of the iris. The margin above and below the iris shall be 0.2R. The upper and lower eyelids shall be masked with a color of 128 while the sclera shall be masked with a color of 200. The boundary between the sclera and eyelids shall be smoothed. See ISO/IEC 19794-6 for further description.

461 Implementation of this function is optional. Implementations that do not support cropping and masking shall return a value of 2. Otherwise, a zero exit status indicates success and the image will be used for matching. If the function returns a value of 8, NIST will debug. Other return values shall indicate an error and the output image will not be used for matching.

Parameters

462	in	*input_iris*	The input iris.
463	out	*output_iris*	The result of the masking and cropping operations. Memory for the raster data will already have been allocated prior to the function call. The amount of memory allocated will be equal to that of the input iris.

Returns

464

465

Return Value	Meaning
0	Success.
2	The implementation does not support this function.
4	Involuntary failure to localize boundaries or perform masking.
6	Elective refusal to produce the output on quality grounds.
8	Cannot parse the input data.
Other	Vendor-defined failure.

466 5 Supporting Data Structures

467 This section describes the data structures used by the API.

468 5.1 CANDIDATE Struct Reference

469 Defines a structure that holds a single candidate.

470 Public Attributes

471 • uint8_t failed

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

473 • uint32_t template_id

474 *Template identifier from the enrollment database.*

475 • double distance_score

Measure of distance between the searched template and the candidate.

- double probability

  Estimate of the probability that the biometric data and candidate belong to different persons.

5.1.1 Detailed Description

Defines a structure that holds a single candidate.

5.1.2 Member Data Documentation

5.1.2.1 uint8_t failed

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

5.1.2.2 uint32_t template_id

Template identifier from the enrollment database.

5.1.2.3 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.

5.1.2.4 double probability

Estimate of the probability that the biometric data and candidate belong to *different* persons.

Stated differently, it shall be the probability that a comparison between two randomly chosen people would produce a distance score less than or equal to the distance score reported above. If the search template is somehow broken, this value shall be set to -1.

5.2 MULTIIRIS Struct Reference

Defines a structure that holds an array of irides for a single person.

Public Attributes

- uint32_t num

  Number of irides.
- ONEIRIS ** irides

  Zero-indexed array of pointers to the irides.

5.2.1 Detailed Description

Defines a structure that holds an array of irides for a single person.

5.2.2 Member Data Documentation

5.2.2.1 uint32_t num

Number of irides.

5.2.2.2 ONEIRIS** irides

Zero-indexed array of pointers to the irides.

5.3 MULTISEGMENTATION Struct Reference

Defines a structure that holds an array of ONESEGMENTATION objects.

Public Attributes

- uint32_t num
  Number of ONESEGMENTATION objects.
- ONESEGMENTATION ** segs
  Zero-indexed array of pointers to ONESEGMENTATION objects.

5.3.1 Detailed Description

Defines a structure that holds an array of ONESEGMENTATION objects.

5.3.2 Member Data Documentation

5.3.2.1 uint32_t num

Number of ONESEGMENTATION objects.

5.3.2.2 ONESEGMENTATION** segs

Zero-indexed array of pointers to ONESEGMENTATION objects.

5.4 ONEIRIS Struct Reference

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

Public Attributes

- uint8_t eye
  Eye label (subject's left or right eye).
- uint16_t image_width
  Image width in pixels.
- uint16_t image_height
  Image height in pixels.
- uint8_t image_type
  Image type integer code.
- uint16_t camera
  The camera sensor ID.
- uint8_t * data
  Pointer to image raster data, 8 bits-per-pixel.

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 uint8_t eye

Eye label (subject's left or right eye).

The eye label information for the OPS-II dataset has proven unreliable and will not be used for testing. This field will always be set to 0, indicating that it is unspecified or unknown.

5.4.2.2 uint16_t image_width

Image width in pixels.

5.4.2.3 uint16_t image_height

Image height in pixels.

5.4.2.4 uint8_t image_type

Image type integer code.

This field has different meaning in IREX IV than it did IREX III. A value of 0 indicates that the image will be either 640x640 or 480x480 with no geometric constraints on the locations of the pupil or iris boundaries. A value of 7 indicates an ISO/IEC 19794-6 Type 7 (cropped and masked) image, the result of a call to convert_raster_to_cropped_and_masked().

5.4.2.5 uint16_t camera

The camera sensor ID.

This field will always be set to 0x0000, meaning that it is either unknown or unspecified.

5.4.2.6 uint8_t* data

Pointer to image raster data, 8 bits-per-pixel.

5.5 ONESEGMENTATION Struct Reference

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

Public Attributes

- double iris_radius

  Iris radius in pixels.
- uint16_t iris_center_x

  x coordinate of iris center.
- uint16_t iris_center_y

  y coordinate of iris center.
- double pupil_radius

Pupil radius in pixels.

- uint16_t pupil_center_x

  x coordinate of pupil center.

- uint16_t pupil_center_y

  y coordinate of iris center.

- uint8_t quality

  Assessment of iris sample quality.

- uint8_t failed

  Indicates whether segmentation of the iris failed (0=success, 1=failed).

5.5.1 Detailed Description

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

5.5.2 Member Data Documentation

5.5.2.1 double iris_radius

Iris radius in pixels.

5.5.2.2 uint16_t iris_center_x

x coordinate of iris center.

5.5.2.3 uint16_t iris_center_y

y coordinate of iris center.

5.5.2.4 double pupil_radius

Pupil radius in pixels.

5.5.2.5 uint16_t pupil_center_x

x coordinate of pupil center.

5.5.2.6 uint16_t pupil_center_y

y coordinate of iris center.

5.5.2.7 uint8_t quality

Assessment of iris sample quality.

Quality is a prediction of how well the sample will perform when matched. 254 indicates quality assessment is unsupported. 255 indicates a failed attempt to assign quality. Otherwise, quality values shall range from 0 to 100, with higher values indicating better quality.

5.5.2.8 uint8_t failed

Indicates whether segmentation of the iris failed (0=success, 1=failed).

6 References

[1] 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. 1, 3

[2] G. Quinn and P. Grother. IREX III supplement I: Failure analysis. Technical report, NIST, 2011. 3, 7

[3] P. Grother, G.W. Quinn, and Jonathan Phillips. Report on the Evaluation of 2D Still-image Face Recognition Algorithms. Technical report, NIST, 2010. 3

[4] 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

[5] R. Bolle, J. Connell, S. Pankanti, N. Ratha, and A. Senior. *Guide to Biometrics*. Springer, 2004. 3

[6] H. Drolon, F. Devaux, A. Descampe, Y. Verschueren, D. Janssens, and B. Macq. OpenJPEG. `http://www.openjpeg.org/`. 5

[7] David Taubman. Kakadu software. `www.kakadusoftware.com`. 5

[8] E. Tabassi, P. Grother, and W. Salamon. IREX - IQCE performance of iris image quality assessment algorithms. Technical report, NIST, 2011. 6

[9] Patrick Grother and Elham Tabassi. Performance of biometric quality measures. *IEEE Trans. Pattern Anal. Mach. Intell*, pages 531–543, 2007. 6