Iris Exchange (IREX) X: Ongoing Evaluation of Iris Recognition

Concept, Evaluation Plan, and API Specification
Identification Track, Version 1.0

George W. Quinn
Information Technology Laboratory, NIST

Questions and Comments to irex@nist.gov

The latest version of this document is available at: https://www.nist.gov/itl/iad/image-group/ongoing-irex

May 17, 2019
Status of this Document

This is the first public version of this document. Comments and questions should be submitted to irex@nist.gov. The latest version of this document can be downloaded from https://www.nist.gov/itl/iad/image-group/ongoing-irex.

Release Notes

The submission procedure, implementation, and runtime requirements for IREX X are similar to those of prior IREX evaluations. However, the API has undergone significant changes.
# Contents

1 Overview ........................................... 1  
1.1 Introduction ........................................... 1  
1.2 Test Dataset ........................................... 1  
1.3 Test Environment ........................................... 1  
1.4 Performance Metrics ........................................... 1  
1.5 Release of Results ........................................... 2  

2 Participation Requirements ........................................... 3  
2.1 Who Can Participate ........................................... 3  
2.2 Submission Procedure ........................................... 3  
2.3 Software Validation ........................................... 3  
2.4 Implementation Requirements ........................................... 3  
2.4.1 Linking ........................................... 3  
2.4.2 Library Name ........................................... 4  
2.4.3 Installation ........................................... 4  
2.4.4 Runtime ........................................... 5  

3 API Specification ........................................... 6  
3.1 Enrollment ........................................... 7  
3.1.1 initialize_enrollment() ........................................... 7  
3.1.2 create_enrollment_template() ........................................... 8  
3.1.3 finalize_enrollment() ........................................... 8  
3.2 Search Templates ........................................... 9  
3.2.1 initialize_search_templates() ........................................... 9  
3.2.2 create_search_template() ........................................... 10  
3.3 Searches ........................................... 11  
3.3.1 initialize_identification() ........................................... 11  
3.3.2 identify() ........................................... 11  
3.4 Supporting Data Structures ........................................... 13  

4 References ........................................... 15  

Participation Agreement ........................................... 16
1 Overview

1.1 Introduction

This document establishes a concept of operations (CONOPS) and application programming interface (API) specification for the Iris Exchange (IREX) X Ongoing Evaluation of Iris Recognition, Identification Track. Administered offline at a NIST’s Biometrics Research Lab (BRL), developers submit their matching software for testing over sequestered iris data. As an ongoing evaluation, developers may submit at any time.

IREX X: Identification Track assesses performance for one-to-many (a.k.a. identification mode) applications. Most flagship deployments of iris recognition operate in one-to-many mode, providing services ranging from prison management, border security, expedited processing, and distribution of resources.

IREX X marks the eighth installment of the IREX umbrella program. IREX was initiated by NIST to support the standardization, development, and deployment of iris-based technology. Although IREX X is the first ongoing IREX evaluation, it does not preclude the possibility that NIST will conduct future one-off evaluations as part of IREX. The latest information on IREX can be found on the IREX website (http://www.nist.gov/itl/iad/ig/irex.cfm).

1.2 Test Dataset

IREX X: Identification Track currently tests over a set of field collected iris images. The samples are pulled from the same source as OPS III (used for performance testing in IREX IX). The samples are collected from various locations over a period of years. Field collected samples tend to suffer more from quality related problems (e.g. motion blur) than samples collected in more controlled laboratory settings. NIST may test over additional datasets going forward.

The dataset is sequestered (i.e. not publicly available). The participants are not allowed to view any of the iris samples and will not be provided with a representative set of iris samples.

Ground Truth Integrity: A hazard with collecting operational data is that ground truth identity labels can be incorrectly assigned due to clerical error. NIST will attempt to correct ground truth errors in its test datasets whenever possible, and only when doing so will not unfairly bias results in favor of specific implementations.

1.3 Test Environment

Testing is performed offline on a secure “air-gapped” network. Hardware specifications for some of the test machines in NIST’s Biometrics Research Laboratory (BRL) are:

- 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).

The test machines have CentOS 7.6 installed. An ISO image of the distribution can be downloaded from nigos.nist.gov (https://nigos.nist.gov/evaluations/CentOS-7-x86_64-Everything-1810.iso).

1.4 Performance Metrics

Performance is assessed for one-to-many matching. One-to-many systems determine whether a user is enrolled in the system and return the user’s identity if found. Although no specific identity claim is made by the user, an implicit claim of enrollment (or lack of enrollment) is still made via the user’s actions. For example, a user presenting his biometric features to an access control system is implicitly claiming to be in enrolled in the system. An offline software test such as IREX X: Identification Track cannot mimic all aspects of a deployed system. Nevertheless, general performance metrics can provide insight into the capabilities of specific biometric technology. Below is a list of performance factors that may be posted to the IREX X website.
Matching Accuracy  One-to-many iris systems are tasked with searching a biometric sample against an enrollment database and returning a list of candidate identities. Associated with each candidate is a comparison score quantifying its degree of dissimilarity to the searched biometric sample. If the comparison score is below a predetermined decision threshold, then the candidate identity is accepted. Expanding on the example of access control, this would grant the user access to the system. If the comparison score for every candidate is above the decision threshold, access is denied.

For any such system, two types of matching errors can occur. The first, a false negative, occurs when the correct candidate for a user is not found. This would result in an authorized user being denied access. The other type of error, a false positive, occurs when an incorrect candidate is returned with a comparison score below the decision threshold. This would result in access being granted for an unauthorized user. Reducing the decision threshold would decrease the rate of false positives but increase the rate of false negatives. Thus, a tradeoff exists between these two core accuracy metrics. Although they do not strictly represent opposing error rates in a binary classification system, accuracy is still presented in the form of Detection Error Trade-off (DET) plots [1]. The precise mathematical definitions of the false negative identification rate (FNIR) and false positive identification rate (FPIR) is presented in several past IREX evaluations [2, 3].

Computation Time  Timing will be assessed as the elapsed real time (as opposed to CPU time) for core biometric operations (e.g. feature extraction, identification). Timing estimates will be computed on an unloaded machine running a single process. The machine’s specifications are described in Section 1.3. NIST may also explore the relationship between search time and the number of entries in the enrollment database, with an eye on whether the relationship is sub-linear.

Accuracy-speed Trade-off  NIST may investigate whether a collective accuracy-speed trade-off exists for submitted implementations. Accuracy-speed trade-off refers to the phenomenon where more accurate implementations take longer to complete their operations.

Template Sizes  NIST may report the size of the proprietary templates generated by the implementation.

Runtime Memory Usage  NIST may monitor runtime memory usage during identification and/or feature extraction.

1.5  Release of Results

NIST will post performance results for each successfully validated implementation to the IREX X webpage (https://www.nist.gov/itl/iad/image-group/ongoing-irex) as soon as they become available. The participant will be notified once posted.
Participation Requirements

2 Participation Requirements

2.1 Who Can Participate

Participation is open to any commercial organization or academic institution that has the ability to implement an iris matching algorithm. There is no charge and participation is open worldwide.

A high-level description of the process is as follows:

1. Mail the completed Participation Agreement to NIST.
2. Wrap your matching software in a C++ software library that adheres to the API specifications and runtime requirements described in the CONOPS document.
3. Submit your implementation to NIST, following the cryptographic protection procedures described in Section 2.2.
4. NIST will validate your submission to ensure its correct operation (Section 2.3).
5. NIST will assess the performance of your implementation and post the results to the IREX X webpage.

Participants only need to complete the Participation Agreement the first time they submit an implementation to IREX X: Identification Track. Participants are permitted to submit one implementation every 3 months.

2.2 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. Implementations shall be submitted to NIST as encrypted gpg files. Encryption shall follow the procedures described at http://www.nist.gov/itl/iad/ig/encrypt.cfm.

If the encrypted implementation is below 20MB, it can be emailed directly to the IREX X Liaison (irex@nist.gov). If the encrypted implementation is above 20MB, it can be provided to NIST as a download from a webserver. NIST shall not be required to register or enroll in any kind of membership before downloading the implementation. Providing NIST with a Google Drive link is acceptable. For security reasons, DropBox links are not currently accepted.

NOTE: NIST will not accept any implementations 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.

2.3 Software Validation

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 validation data along with instructions are downloadable from the IREX X webpage: https://www.nist.gov/itl/iad/image-group/ongoing-irex.

2.4 Implementation Requirements

2.4.1 Linking

Participants shall submit their implementations as pre-compiled and linkable libraries. Dynamic libraries are permitted, but static ones are preferred. Participants shall not provide any source code. Header
Participation Requirements

files should not be necessary, but if provided, should not contain intellectual property of the company nor any material that is otherwise proprietary.

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. Thus, libraries must export their functions according to that compilers naming convention. The functions that must be exported are defined in "irex.h" (found in the validation package) and described in Section 3. The software libraries must be 64-bit. 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_00 -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.). NIST’s test machines will have CentOS 7.6 installed, which can be downloaded from nigos.nist.gov (https://nigos.nist.gov/evaluations/CentOS-7-x86_64-Everything-1810.iso).

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.

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 new implementations.

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.

2.4.2 Library Name

All submitted libraries shall adhere to the naming convention described in Table 1. Additional dynamic or shared library files may be submitted that support this core library.

<table>
<thead>
<tr>
<th>Format:</th>
<th>libIREX_participantName_index.ext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part:</td>
<td>libIREX</td>
</tr>
<tr>
<td>Description: Fixed for all submissions.</td>
<td>A short name for the participant (e.g. &quot;thebes&quot;).</td>
</tr>
<tr>
<td>Example:</td>
<td>libIREX_thebes_00.a</td>
</tr>
</tbody>
</table>

2.4.3 Installation

**Installation Must be Simple** Installation shall require the simple copying and/or decompression 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.

**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.

**Sufficient Documentation Must be Provided** Documentation should be provided for all (non-zero) participant-defined error or warning return codes.
Participation Requirements

**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 any given submission shall not be more than a gigabyte.

### 2.4.4 Runtime

**Single-threaded Requirement** Implementations must run in single-threaded mode.

**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.

**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.

**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 explicitly 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.

**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.

**Minimum Speed Requirements** The implementations shall perform operations within the time constraints specified by Table 2. These time limits apply to the function call invocations defined in Section 3 on a Dell M910 system described in Section 1.3. 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 is generated from a single iris image.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Timing Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation of a verification template from a single 640x480 pixel image</td>
<td>1000 ms</td>
</tr>
<tr>
<td>Creation of an enrollment template from a single 640x480 pixel image</td>
<td>1000 ms</td>
</tr>
<tr>
<td>Search duration on a database of one million templates</td>
<td>20 000 ms</td>
</tr>
</tbody>
</table>

**Template Size** Templates should not require more than 100 KB of persistent storage per iris image. Participants should inform NIST if their implementations require more than 100 KB of persistent storage.
3 API Specification

The design of this API reflects the following testing objectives:

• Support black-box testing.
• Support distributed processing.
• Support graceful and informative failure recovery.
• Support the ability to collect performance statistics.

Submitted library files must export and properly implement the functions defined in this section. Testing will proceed in three phases: (1) construction of the enrollment database, (2) creation of search templates, (3) searching against the enrollment database. This basic program flow is further described in Table 3.

The API specifications in this section were auto-generated from the "irex.h" header file distributed with the validation package.

Table 3: Program Flow

<table>
<thead>
<tr>
<th>Stage</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enrollment</td>
<td>initialize_enrollment()</td>
</tr>
<tr>
<td></td>
<td>Allows the implementation to perform initialization procedures.</td>
</tr>
<tr>
<td></td>
<td>Provides the implementation with advance knowledge of the number of images that will be enrolled as well as read-only access to a participant-supplied configuration directory.</td>
</tr>
<tr>
<td></td>
<td>create_enrollment_template()</td>
</tr>
<tr>
<td></td>
<td>Creates an enrollment template from one or more iris images.</td>
</tr>
<tr>
<td></td>
<td>The implementation must be able to handle multiple calls to this function from multiple instances of the calling application.</td>
</tr>
<tr>
<td></td>
<td>finalize_enrollment()</td>
</tr>
<tr>
<td></td>
<td>Constructs an enrollment database from all enrollment templates.</td>
</tr>
<tr>
<td></td>
<td>This function allows post-enrollment book-keeping, normalization, and other statistical processing of the templates. The contents of the enrollment directory should be populated with everything necessary to perform searches against it.</td>
</tr>
<tr>
<td>Pre-search</td>
<td>initialize_search_template()</td>
</tr>
<tr>
<td></td>
<td>Prepares the implementation for the creation of search templates.</td>
</tr>
<tr>
<td></td>
<td>The implementation is allowed read-only access to the enrollment directory at this stage.</td>
</tr>
<tr>
<td></td>
<td>create_search_templates()</td>
</tr>
<tr>
<td></td>
<td>Creates a search template from one or more iris images.</td>
</tr>
<tr>
<td></td>
<td>The implementation must be able to handle multiple calls to this function from multiple instances of the calling application.</td>
</tr>
</tbody>
</table>
### 3.1 Enrollment

- **int32_t initialize_enrollment (const std::string &config_dir="")**
  
  *Initialization function, called once prior to one or more calls to create_enrollment_template().*

- **int32_t create_enrollment_template (const std::vector< iris_image > &irides, std::vector< uint8_t > &enrollment_template)**
  
  *Generates an enrollment template from a vector of iris images.*

- **int32_t finalize_enrollment (const std::string &enrollment_dir, const std::vector< enrollment_entry > &enrollment_templates)**
  
  *Finalization function, used to construct an enrollment database to search against.*

#### 3.1.1 initialize_enrollment()

```cpp
int32_t IREX::TRACK1::initialize_enrollment (  
    const std::string & config_dir = "" )
```

*Initialization function, called once prior to one or more calls to create_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.

NiST will debug if a call to this function returns a non-zero value.

If participants wish to use the `config_dir` parameter, they must state as much in documentation accompanying their submission.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>config_dir</th>
<th>An (optional) absolute path to a read-only directory containing participant-supplied configuration parameters and/or runtime data files.</th>
</tr>
</thead>
</table>

---

**initialize_identification()**

*Prepares the implementation for searches against the enrollment database.*

*The implementation is allowed to read data (e.g. templates) from the enrollment directory and load them into memory.*

**identify()**

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

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>5</td>
<td>The configuration data is missing, unreadable, or in an unexpected format.</td>
</tr>
<tr>
<td>Other</td>
<td>Participant-defined failure.</td>
</tr>
</tbody>
</table>

### 3.1.2 create_enrollment_template()

```cpp
int32_t IREX::TRACK1::create_enrollment_template(
    const std::vector< iris_image > & irides,
    std::vector< uint8_t > & enrollment_template )
```

Generates an enrollment template from a vector of iris images.

The implementation must be able to handle iris images having arbitrary dimensions, although NIST does not expect to test over images that are not 640x480 or 640x512.

If the return value is less than 100, the calling function will store the template and later pass it to `finalize_enrollment()`. The template is allowed to be zero bytes in size. If a return value of 100 or greater is returned, NIST will attempt to debug, contacting the participant if necessary.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>irides</th>
<th>The iris images from which to create the template.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>enrollment_template</td>
<td>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 <code>finalize_enrollment()</code> and possibly store it temporarily.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Failure to find the iris.</td>
</tr>
<tr>
<td>2</td>
<td>The input data is in an incorrect or unrecognized format.</td>
</tr>
<tr>
<td>10 ≤ ret &lt; 100</td>
<td>Participant-defined soft failure. Soft failures simply allow the participant to provide more descriptive output on why a useful template could not be created.</td>
</tr>
<tr>
<td>100 ≤ ret</td>
<td>Participant-defined hard failure. Hard failures should indicate unexpected events that might compromise the validity of test results.</td>
</tr>
</tbody>
</table>

### 3.1.3 finalize_enrollment()

```cpp
int32_t IREX::TRACK1::finalize_enrollment(
    const std::string & enrollment_dir,
    const std::vector< enrollment_entry > & enrollment_templates )
```

Finalization function, used to construct an enrollment database to search against.
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, although subsequent calls should probably not do anything.

The NIST test harness will never knowingly enroll the same iris under two different identifiers. NIST will debug if a call to this function returns a non-zero value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enrollment_dir</td>
<td>An absolute path to the top-level directory in which the enrollment database will reside. The implementation will have read and write access to this directory.</td>
</tr>
<tr>
<td>enrollment_templates</td>
<td>The vector of enrollment templates that should comprise the enrollment database. The implementation should store these templates in the enrollment directory.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>2</td>
<td>One or more enrollment templates are in an incorrect format.</td>
</tr>
<tr>
<td>4</td>
<td>An operation on the enrollment directory failed.</td>
</tr>
<tr>
<td>Other</td>
<td>Participant-defined failure.</td>
</tr>
</tbody>
</table>

3.2 Search Templates

- int32_t initialize_search_templates (const std::string & enrollment_dir, const std::string & config dir = "")
  Initialization function, to be called once prior to one or more calls to create_search_template().

- int32_t create_search_template (const std::vector< iris_image > & irides, std::vector< uint8_t > & search_template)
  Generates a search template from a vector of iris samples.

3.2.1 initialize_search_templates()

int32_t IREX::TRACK1::initialize_search_templates (  
    const std::string & enrollment_dir,  
    const std::string & config_dir = ""  )

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

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

NIST will debug if a call to this function returns a non-zero value.

If participants wish to use the config_dir parameter, they must state as much in documentation accompanying their submission.
API Specification

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>enrollment_dir</th>
<th>The top-level directory in which the enrollment data was placed when finalize_enrollment() was called.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>config_dir</td>
<td>An (optional) absolute path to a read-only directory containing participant-supplied configuration parameters and/or runtime data files.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>4</td>
<td>An operation on the enrollment directory failed.</td>
</tr>
<tr>
<td>5</td>
<td>The configuration data is missing, unreadable, or in an unexpected format.</td>
</tr>
<tr>
<td>Other</td>
<td>Participant-defined failure.</td>
</tr>
</tbody>
</table>

3.2.2 create_search_template()

```cpp
int32_t IREX::TRACK1::create_search_template (  
    const std::vector< iris_image > & irides,  
    std::vector< uint8_t > & search_template )
```

Generates a search template from a vector of iris samples.

The implementation must be able to handle iris images having arbitrary dimensions, although NIST does not expect to test over images that are not 640x480 or 640x512.

If the return value is less than 100, the calling function will store the template and later pass it to identify(). The template is allowed to be zero bytes in size. If a return value of 100 or greater is returned, NIST will attempt to debug, contacting the participant if necessary.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>irides</th>
<th>The iris samples from which to generate the template.</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>search_template</td>
<td>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() and possibly store it temporarily.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>1</td>
<td>Failure to find the iris.</td>
</tr>
<tr>
<td>2</td>
<td>The input data is in an incorrect or unrecognized format.</td>
</tr>
<tr>
<td>10 ≤ ret &lt; 100</td>
<td>Participant-defined soft failure. Soft failures simply allow the participant to provide more descriptive output on why a useful template could not be created.</td>
</tr>
<tr>
<td>100 ≤ ret</td>
<td>Participant-defined hard failure. Hard failures should indicate unexpected events that might compromise the validity of test results.</td>
</tr>
</tbody>
</table>
3.3 Searches

- int32_t \texttt{initialize\_identification} (const std::string &\texttt{enrollment\_dir}, const std::string &\texttt{config\_dir}="")
  
  Initialization function, to be called once prior to one or more calls to \texttt{search\_template()}.

- int32_t \texttt{identify} (const std::vector<uint8_t> &\texttt{search\_template}, const uint32_t \texttt{num\_candidates},
  std::vector<\texttt{candidate}> &\texttt{candidates})
  
  Searches a template against the enrollment database and returns a list of candidates.

### 3.3.1 \texttt{initialize\_identification()}

```cpp
int32_t IREX::TRACK1::initialize\_identification(
    const std::string & enrollment\_dir,
    const std::string & config\_dir = ""
) {
    // Initialization function, to be called once prior to one or more calls to search\_template().
    // The function may read data (e.g. templates) from the enrollment directory and load them into memory.
    // NIST will debug if a call to this function returns a non-zero value.
    // If participants wish to use the \texttt{config\_dir} parameter, they must state as much in documentation accompanying their submission.

    // Parameters
    // in \texttt{enrollment\_dir} - The top-level directory in which the enrollment data was placed when finalize\_enrollment() was called.
    // in \texttt{config\_dir} - An (optional) absolute path to a \textit{read-only} directory containing participant-supplied configuration parameters and/or runtime data files.

    // Returns
    // Return Value | Description
    // 0 | Success.
    // 4 | An operation on the enrollment directory failed.
    // 5 | The configuration data is missing, unreadable, or in an unexpected format.
    // Other | Participant-defined hard failure.
```

### 3.3.2 \texttt{identify()}

```cpp
int32_t IREX::TRACK1::identify(
    const std::vector<uint8_t> & search\_template,
    const uint32_t num\_candidates,
    std::vector<\texttt{candidate}> & candidates ) {
    // Searches a template against the enrollment database and returns a list of candidates.
    // The implementation must be able to handle search\_templates even when the calling function, create\_search\_template(), returned a non-zero value.
    // 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.
```
API Specification

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>search_template</th>
<th>A template generated by a call to create_search_template().</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>num_candidates</td>
<td>The length of the candidate list array.</td>
</tr>
<tr>
<td>out</td>
<td>candidates</td>
<td>An array (of length num_candidates) of pointers to candidates. Each candidate shall be populated by the implementation and shall be sorted in ascending order of distance score (i.e. the most similar entry shall appear first).</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success.</td>
</tr>
<tr>
<td>3</td>
<td>The input template does not contain matchable information.</td>
</tr>
<tr>
<td>10 ≤ ret &lt; 100</td>
<td>Participant-defined soft failure. Soft failures simply allow the participant to provide more descriptive output on why the search could not produce valid results.</td>
</tr>
<tr>
<td>100 ≤ ret</td>
<td>Participant-defined hard failure. Hard failures should indicate unexpected events that might compromise the validity of test results.</td>
</tr>
</tbody>
</table>
3.4 Supporting Data Structures

This section describes the data structures used by the API.

**point Struct Reference**  
Specifies a coordinate in an image.

**Public Attributes**
- `uint16_t x`  
  x-coordinate (0 == leftmost).
- `uint16_t y`  
  y-coordinate (0 == topmost).

**iris_image Struct Reference**  
Stores image data for an iris.

**Public Attributes**
- `uint16_t width`  
  Image width in pixels.
- `uint16_t height`  
  Image height in pixels.
- `uint8_t *data`  
  Pointer to image raster data (RGBRGBRGB... for 24-bit images).
- `uint8_t depth`  
  Bit depth, 8 for grayscale or 24 for RGB visible spectrum images.
- `uint8_t eye`  
  The eye label (0 = undefined, 1 = right eye, 2 = left eye).
- `uint16_t wavelength`  
  Wavelength in nm at which image was acquired (0 == unspecified).

**candidate Struct Reference**  
Stores a single candidate returned by search().

**Public Attributes**
- `bool valid`  
  Indicates whether the candidate is valid.
- `uint32_t id`  
  Unique template identifier database.
- `double distance`  
  Measure of dissimilarity (aka distance) between the searched template and the candidate.

**Member Data Documentation**

**valid**  
bool valid
Indicates whether the candidate is valid.
If this value is non-zero, the values in `template_id` and `distance` will be ignored.

**id**  
uint32_t id
Unique template identifier database.
One of the template identifiers originally passed to `finalize_enrollment()`.

**distance**  
double distance
Measure of dissimilarity (aka distance) between the searched template and the candidate.
This value must be non-negative to be considered valid.

**enrollment_entry Struct Reference**  
An enrollment template with associated unique identifier.
Public Attributes

- `std::vector<uint8_t> enrollment_template`
  *The enrollment template.*

- `uint32_t id`
  *Unique identifier assigned to this template.*


4 References


Application to Participate in Iris Exchange (IREX) X

1. Who Can participate

1.1. Iris recognition researchers and developers from industry, research institutions, and academia are eligible to participate in Iris Exchange X: Ongoing Evaluation of Iris Recognition - hereafter referred to as IREX X.

1.2. Signatories to this document, Application to Participate in Iris Exchange X (Agreement), acknowledge that they understand the results of this test will be published with attribution to their Organization.

2. How to Participate

2.1. To participate, an Organization must complete this form and mail it to the designated address. The
   • Responsible Party is an individual with the authority to commit the Organization to the terms in this Agreement.
   • Point of Contact is an individual within the Organization with detailed knowledge of the submitted software Implementation.
   • The Responsible Party and Point of Contact may be the same person.

2.2. Upon receipt of the signed agreement, the Organization will be classified as a Participant in IREX X.

2.3. The Participant shall submit an iris recognition Implementation to NIST that is conformant with the Concept, Evaluation Plan, and API Specification associated with the test, available on the IREX X website https://www.nist.gov/itl/iad/image-group/ongoing-irex.

2.4. Transmission of Implementations to NIST must follow the submission procedures detailed in the CONOPS and API specification document. This involves the encryption of all proprietary software submitted to NIST. The encryption procedure can be found at http://www.nist.gov/itl/iad/ig/encrypt.cfm.

2.5. The Participant may withdraw from IREX X at any time before their Implementation is received by NIST, without their participation or withdrawal being documented on the IREX X website.

2.6. Organizations only need to mail this Agreement the first time they submit an Implementation to IREX X. Organizations need not mail a new application each time they submit an Implementation.

3. Point of Contact

3.1. The IREX X Liaison is the U.S. Government point of contact for IREX X. All questions should be directed to irex@nist.gov, which will be received by the IREX X Liaison and other IREX personnel.

4. Release of Results

4.1. After successful completion of testing, NIST will publish the results along with the Organization's name to the IREX X website.

4.2. The Point of Contact will be notified once results are posted.

4.3. References to results in IREX X shall be accompanied by the phrase “Results presented in IREX do not constitute an endorsement, recommendation, or favoring of any particular system, product, or service by the U.S. Government.” A reference to the IREX website is also requested.

4.4. All published results are provided in good faith and NIST takes no legal responsibility for the accuracy of the information provided. In the event that errors are discovered, NIST will re-run prior tests to resolve those errors.
5. **Legal**

5.1. To the extent allowable by Federal law, NIST agrees to protect Implementations as business proprietary information by using same degree of care, but no less than a reasonable degree of care, to prevent the unauthorized use, dissemination or publication of the Proprietary Information as NIST uses to protect its own confidential information of a like nature. NIST agrees not to transfer any Implementations to any third party without the prior written permission of the Participant.

5.2. These provisions are consistent with and do not supersede, conflict with, or otherwise alter the employee obligations, rights, or liabilities created by existing statute or Executive order relating to (1) classified information, (2) communications to Congress, (3) the reporting to an Inspector General of a violation of any law, rule, or regulation, or mismanagement, a gross waste of funds, an abuse of authority, or a substantial and specific danger to public health or safety, or (4) any other whistleblower protection. The definitions, requirements, obligations, rights, sanctions, and liabilities created by controlling Executive orders and statutory provisions are incorporated into this Agreement and are controlling.

5.3. Any data obtained during IREX X, as well as any documentation required by NIST from the Participant (except the Implementation), becomes the property of NIST. The Participant will not acquire a proprietary interest in the data and/or submitted documentation. Any data or documentation that qualifies as business proprietary information, as defined by the Freedom of Information Act (5 USC Section 552) will be treated as confidential in accordance with Section 2, and will only be used for the purposes of IREX testing.

5.4. Participant agrees that they will not file any IREX X related claim against IREX 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.

5.5. The U.S. Government is not bound or obligated to follow any recommendations requested by the Participant. The U.S. Government is not bound, nor obligated in any way, to give any special consideration to the Participant on future contracts.

5.6. By signing this Agreement, the Participant acknowledges that they understand any test details and/or modifications that are provided in the IREX X website supersede the information on this Agreement.
Application to Participate in Iris Exchange (IREX) X

Please mail this completed form, along with attached business cards from both signing parties, to:

IREX X Test Liaison
National Institute of Standards and Technology
Information Access Division (774.03)
100 Bureau Drive, Mailstop 8940
Gaithersburg, MD 20899-8940

This Application must be mailed. Scanned documents sent as email attachments will not be accepted. Please send an email to irex@nist.gov once you have mailed your application. Upon receipt, NIST will respond with a confirmation.

The Participant must provide a Public Key per the instructions for the transmission of encrypted content to NIST at http://www.nist.gov/itl/iad/ig/encrypt.cfm. Alternatively, the Participant may write "faxed" in the box below and fax the public key to the IREX X Test Liaison at (301) 975-5287.

<table>
<thead>
<tr>
<th>Participant Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NIST Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>2284 D56A F01B D81D 6F2A 16BE B648 74EA DBA5 F3A8</td>
</tr>
</tbody>
</table>

With my signature, I hereby request consideration as a Participant in IREX X, and I authorize my Organization to participate 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   Email Address               Date

______________________________  ______________________________  ____________
Signature of Point of Contact    Email Address               Date