Face In Video Evaluation (FIVE)
Concept, Evaluation Plan, and API
Version 0.4

Patrick Grother and Mei Ngan

Image Group
Information Access Division
Information Technology Laboratory

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

October 3, 2014

Timeline of the FIVE Evaluation

Phase	Date	External actions, deadlines
Phase 0	2014-07-15	Web site up, announce schedule
	2014-08-15	First draft Evaluation Plan and API
	2014-08-31	Public comments on first drafts due
	2014-10-01	Second draft Evaluation Plan and API
	2014-10-15	Public comments on second drafts due
	2014-10-30	Third draft Evaluation Plan and API. Draft five.h available.
	2014-11-07	Public comments on third drafts due
	2014-11-10	Final Evaluation Plan and API available. Final five.h available
	2014-11-10	FIVE validation package available
	2014-11-17	Updates to FIVE validation package as necessary
Phase 1	2014-11-17	Opening of Phase 1 submission period
	2015-02-08	Deadline for submission for inclusion of results in first interim report card
	2015-03-28	First interim report card released to submitting participants
Phase 2	2015-04-01	Opening of Phase 2 submission period
	2015-06-05	Deadline for submission for inclusion of results in second interim report card.
	2015-07-30	Second interim report card released to submitting participants
Phase 3	2015-08-01	Opening of Phase 3
	2015-10-05	Deadline for submission of algorithms to Phase 3

November 2014

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

December 2014

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

January 2015

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

February 2015

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28

March 2015

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

April 2015

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

May 2015

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

June 2015

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

July 2015

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

August 2015

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

September 2015

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Major API Changes since FRVT 2013 Class V

The header/source files for the API will be made available to implementers at http://nigos.nist.gov:8080/five.

— The structures ONEFACE (see Table 11) and MULTIFACE (see Table 12) have been changed to classes.

— The MULTIFACE class contains a new "description" member variable and valid values are specified in Table 10.

— The labels for describing types of still images have been updated (see Table 9).

— The ONEVIDEO (see Table 14) class contains a new "peopleDensity" member variable and valid values are specified in Table 13.

Table of Contents

List of Figures

List of Tables

Acknowledgements

— The authors are grateful to the experts who made extensive comments on the first version of this document.

Project History

— 2012 – 2014 – The FRVT 2013 program included a video track (class V) that evaluated face recognition from video. The FIVE program supersedes the FRVT work but proceeds in an almost identical manner.

— August 15, 2014 - Release of first public draft of the Face In Video Evaluation (FIVE) – Concept, Evaluation Plan and API v0.1.

Terms and definitions

The abbreviations and acronyms of Table 1 are used in many parts of this document.

Table 1 – Abbreviations

FNIR	False negative identification rate
FPIR	False positive identification rate
FIVE	NIST’s Face In Video Evaluation program
FRVT	NIST’s Face Recognition Vendor Test program
FTA	Failure to acquire a search sample
FTE	Failure to extract features from an enrollment image
DET	Detection error tradeoff characteristic: For identification this is a plot of FNIR vs. FPIR.
INCITS	InterNational Committee on Information Technology Standards
ISO/IEC 19794	ISO/IEC 19794-5: Information technology — Biometric data interchange formats — Part 5:Face image data. First edition: 2005-06-15. (See Bibliography entry).
MBE	NIST's Multiple Biometric Evaluation program
NIST	National Institute of Standards and Technology
SDK	The term Software Development Kit refers to any library software submitted to NIST. This is used synonymously with the terms "implementation" and "implementation under test".

1. FIVE

1.1. Scope

The Face In Video Evaluation (FIVE) is being conducted to assess the capability of face recognition algorithms to correctly identify or ignore persons appearing in video sequences – i.e. the open-set identification problem. Both comparative and absolute accuracy measures are of interest, given the goals to determine which algorithms are most effective and whether any are viable for the following primary operational use-cases:

1. High volume screening of persons in the crowded spaces (e.g. an airport)
2. Low volume forensic examination of footage from a crime scene (e.g. a convenience store)
3. Persons in business meetings (e.g. for video-conferencing)
4. Persons appearing in television footage

These applications differ in their tolerance of false positives, whether a human examiner will review outputs, the prior probabilities of mate vs. non-mate presence, and the cost of recognition errors.

Out of scope: Areas that are out of scope for this evaluation and will not be studied include: gait, iris and voice recognition; recognition across multiple views (e.g. via stereoscopic techniques); tracking across sequential cameras (re-identification); anomaly detection; detection of evasion.

This document establishes a concept of operations and an application programming interface (API) for evaluation of face recognition in video implementations submitted to NIST's Face In Video Evaluation. See http://www.nist.gov/itl/iad/ig/five.cfm for all FIVE documentation.

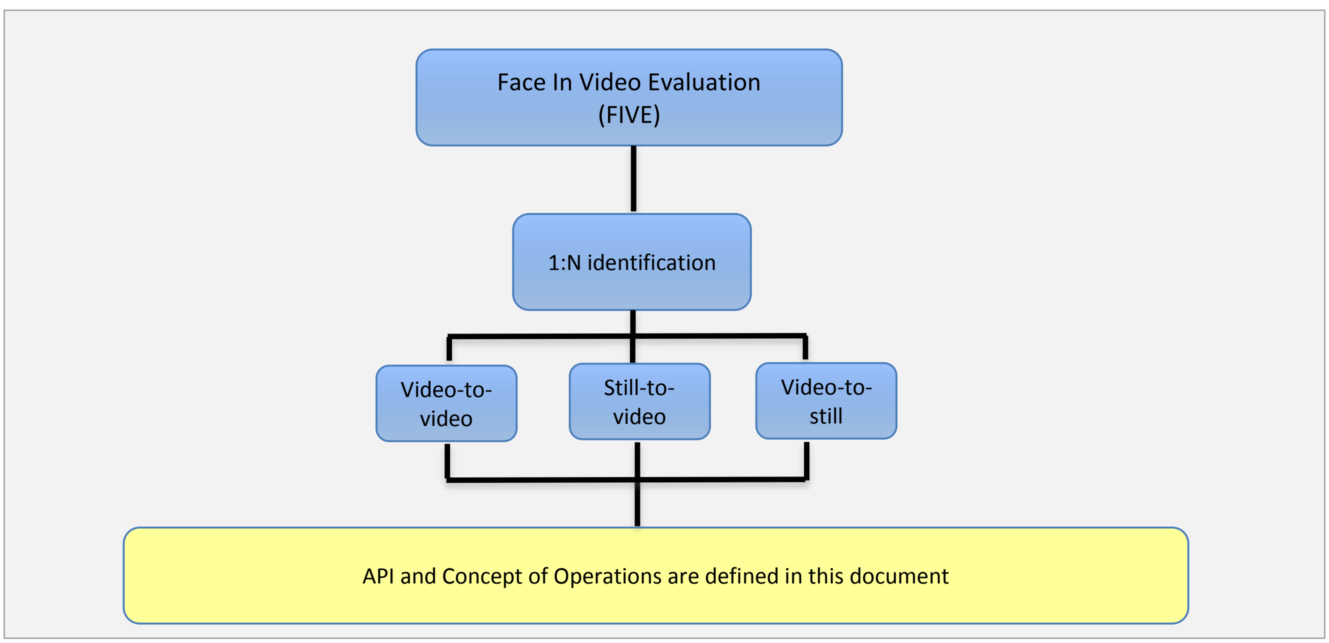


Figure 1 – Organization and documentation of the FIVE

1.2. Audience

Universities and commercial entities with capabilities in detection and identification of faces in video sequences are invited to participate in the FIVE Video test.

Organizations will need to implement the API defined in this document. Participation is open worldwide. There is no charge for participation. While NIST intends to evaluate technologies that could be readily made operational, the test is also open to experimental, prototype and other technologies.

1.3. Market drivers

This test is intended to support a plural marketplace of face recognition in video systems. There is considerable interest in the potential use of face recognition for identification of persons in videos.

1.4. Offline testing

While this set of tests is intended as much as possible to mimic operational reality, this remains an offline test executed on databases of images. The intent is to assess the core algorithmic capability of face recognition in video algorithms. This test will be conducted purely offline - it does not include a live human-presents-to-camera component. Offline testing is attractive because it allows uniform, fair, repeatable, and efficient evaluation of the underlying technologies. Testing of implementations under a fixed API allows for a detailed set of performance related parameters to be measured.

1.5. Phased testing

To support research and development efforts, this testing activity will embed multiple rounds of testing. These test rounds are intended to support improved performance. Once the test commences, NIST will evaluate implementations on a first-come-first-served basis and will return results to providers as expeditiously as possible. Providers may submit revised SDKs to NIST only after NIST provides results for the prior SDK and invites further submission. The frequency with which a provider may submit SDKs to NIST will depend on the times needed for developer preparation, transmission to NIST, validation, execution and scoring at NIST, and developer review and decision processes.

For the schedule and number of SDKs of each class that may be submitted, see sections 1.10 and 1.11.

1.6. Interim reports

The performance of each SDK will be reported in a "score-card". This will be provided to the participant. While the score cards may be used by the provider for arbitrary purposes, they are intended to facilitate development. Score cards will

– be machine generated (i.e. scripted),
– be provided to participants with identification of their implementation,
– include timing, accuracy and other performance results,
– include results from other implementations, but will not identify the other providers,
– be expanded and modified as revised implementations are tested, and as analyses are implemented,
– be generated and released asynchronously with SDK submissions,
– be produced independently of the other status of other providers’ implementations,
– be regenerated on-the-fly, usually whenever any implementation completes testing, or when new analysis is added.

NIST does not intend to release these interim test reports publicly. NIST may release such information to the U.S. Government test sponsors. While these reports are not intended to be made public, NIST can only request that agencies not release this content.

1.7. Final reports

NIST will publish one or more final public reports. NIST may also

– publish additional supplementary reports (typically as numbered NIST Interagency Reports),
– publish in other academic journals,
– present results at conferences and workshops (typically PowerPoint).

Our intention is that the final test reports will publish results for the best-performing implementation from each participant. Because "best" is ill-defined (accuracy vs. time vs. template size, for example), the published reports may include results for other implementations. The intention is to report results for the most capable implementations (see section 1.13, on metrics). Other results may be included (e.g. in appendices) to show, for example, examples of progress or tradeoffs. IMPORTANT: Results will be attributed to the providers.

1.8. Application scenarios

This test will include one-to-many identification tests for video sequences. As described in Table 2, the test is intended to represent identification applications for face recognition in video.

Table 2 – Subtests supported under the FIVE activity

#		Video-to-Video	Video-to-Still	Still-to-Video
1.	Aspect	1:N identification of video-to-video	1:N identification of video-to-still	1:N identification of still-to-video
2.	Enrollment dataset	N enrolled video sequences	N enrolled stills	N enrolled video sequences
3.	Prior NIST test references	Equivalent to 1 to N matching in [FRVT 2013]		
4.	Example application	Open-set identification against a central database, e.g. a search of a wanted criminal through a live-video surveillance system at an airport who may attempt to flee the country		
5.	Score or feature space normalization support	Any score or feature based statistical normalization techniques-are applied against enrollment database		
6.	Intended number of subjects	Expected $O(10^2)$ - $O(10^4)$		
7.	Number of images per individual	N/A	Variable, see section 1.12.	Variable, see section 1.12.

1.9. Image source labels

NIST may mix images from different sources in an enrollment set. For example, NIST could combine frontal images and images with varying poses into a single enrollment dataset. For this reason, in the data structure defined in clause 2.3.3, each image is accompanied by a "label" which identifies the set-membership images. Legal values for labels are in clause 2.3.2.

1.10. Rules for participation

A participant must properly follow, complete and submit a participation agreement (see Annex A). This must be done once, not before November 17, 2014. It is not necessary to do this for each submitted SDK. All submitted SDKs must meet the API requirements as detailed in section 3.

1.11. Number and schedule of submissions

The test is conducted in three phases, as scheduled on page 2. The maximum total (i.e. cumulative) number of submissions is regulated in Table 3.

Table 3 – Cumulative total number of algorithms

#	Phase 1	Total over Phases 1 + 2	Total over Phases 1 + 2 + 3
Cumulative total number of submissions	2	3	5 if at least 1 was successfully executed by end Phase 2 2 if zero had been successfully executed by end Phase 2

The numbers above may be increased as resources allow.

NIST cannot conduct surveys over runtime parameters because NIST must limit the extent to which participants are able to train on the test data.

1.12. Use of multiple images per person

Some of the proposed datasets includes K > 2 images per person for some persons. For video-to-still recognition in this test, NIST will enroll K ≥ 1 images under each identity. For still-to-video, the probe will consist of K ≥ 1 images. Normally the probe will consist of a single image, but NIST may examine the case that it could consist of multiple images. The method by which the face recognition implementation exploits multiple images is not regulated: The test seeks to evaluate developer provided technology for multi-presentation fusion. This departs from some prior NIST tests in which NIST executed fusion algorithms (e.g. [FRVT2002b]), and sum score fusion, for example, [MINEX]).

This document defines a template to be the result of applying feature extraction to a set of K ≥ 1 images or K ≥ 1 video frames. That is, a template contains the features extracted from one or more images or video frames, not generally just one. An SDK might internally fuse K feature sets into a single representation or maintain them separately - In any case the resulting proprietary template is contained in a contiguous block of data. All identification functions operate on such multi-image or multi-frame templates.

The number of images per person will depend on the application area:

— In civil identity credentialing (e.g. passports, driving licenses) the images will be acquired approximately uniformly over time (e.g. five years for a Canadian passport). While the distribution of dates for such images of a person might be assumed uniform, a number of factors might undermine this assumption[1].

— In criminal applications the number of images would depend on the number of arrests[2]. The distribution of dates for arrest records for a person (i.e. the recidivism distribution) has been modeled using the exponential distribution, but is recognized to be more complicated. NIST currently estimates that the number of images will never exceed 100.

1.13. Core accuracy metrics

For identification testing, the test will target open-universe applications such as benefits-fraud and watch-lists. It will not address the closed-set task because it is operationally uncommon.

While some one-to-many applications operate with purely rank-based metrics, this test will primarily target score-based identification metrics. Metrics are defined in Table 4. The analysis will survey over various rank and thresholds[3]. Plots of the two error rates, parametric on threshold, will be the primary reporting mechanism.

Table 4 – Summary of accuracy metrics

Application	Metric		
1:N Identification (Video-to-Still)	FPIR	=	The rate at which unknown subjects are incorrectly associated with any of N enrolled identities. The association will be parameterized on a continuous threshold T.
	FNIR	=	The rate at which known subjects are incorrectly not associated with the correct enrolled identities. The association will be parameterized on a continuous threshold T, and a candidate rank, R.

FPIR will be estimated using probe images or video clips for which there is no enrolled mate. The stability of FPIR at a fixed threshold under changes to image properties or demographics will be reported.

NIST will extend the analysis in other areas, with other metrics, and in response to the experimental data and results.

[1] For example, a person might skip applying for a passport for one cycle (letting it expire). In addition, a person might submit identical images (from the same photography session) to consecutive passport applications at five year intervals.

[2] A number of distributions have been considered to model recidivism, see ``Random parameter stochastic process models of criminal careers.'' In Blumstein, Cohen, Roth & Visher (Eds.), Criminal Careers and Career Criminals, Washington, D.C.: National Academy of Sciences Press, 1986.

[3] Threshold and rank limits are established operationally to limit human labor requirements: One the one side, in a low volume forensic application e.g. investigation of video collected in a convenience store hold-up, or in looking at videos of passengers dis-embarking flights to document an asylum claim, an examiner might be willing to adjudicate R >> 1 candidates with threshold, T = 0. At the other end, a high volume watch-list application in which crowded airport concourses are surveilled for bad actors, a high threshold would be used to limit false positive outcomes. In that case, candidate lists will often have zero length. NIST will report metrics appropriate to the "human-automated" hybrid application, and the "lights-out" hits-are-rare use case.

1.14. Generalized accuracy metrics

Under the ISO/IEC 19795-1 biometric testing and reporting standard, a test must account for "failure to acquire" (FTA) and "failure to enroll" (FTE) events (e.g. elective refusal to make a template, or fatal errors). The way these are treated is application-dependent.

For identification, the appropriate metrics reported in FIVE will be generalized to include FTA and FTE events.

1.15. Reporting template size

Because template size is influential on storage requirements and computational efficiency, this API supports measurement of template size. NIST will report statistics on the actual sizes of templates produced by face recognition implementations submitted to FIVE. NIST may report statistics on runtime memory usage. Template sizes were reported in the FRVT 2013 test[4], IREX III test[5], and the MBE-STILL 2010 test[6].

1.16. Reporting computational efficiency

As with other tests, NIST will compute and report recognition accuracy. In addition, NIST will also report timing statistics for all core functions of the submitted SDK implementations. This includes feature extraction and 1:N recognition. For an example of how efficiency can be reported, see the final report of the FRVT 2013 test, IREX III test, and the MBE-STILL 2010 test.

1.17. Exploring the accuracy-speed trade-space

NIST will explore the accuracy vs. speed tradeoff for face recognition algorithms running on a fixed platform. NIST will report both accuracy and speed of the implementations tested. While NIST cannot force submission of "fast vs. slow" variants, participants may choose to submit variants on some other axis (e.g. "experimental vs. mature") implementations. NIST encourages “fast-less-accurate vs. slow-more-accurate” with a factor of three between the speed of the fast and slow versions.

1.18. Hardware specification

NIST intends to support high performance by specifying the runtime hardware beforehand. There are several types of computer blades that may be used in the testing. The blades are labeled as Dell M905, M910, M605, and M610. The following list gives some details about the hardware of each blade type:

- Dell M605 - Dual Intel Xeon E5405 2 GHz CPUs (4 cores each)
- Dell M905 - Quad AMD Opteron 8376HE 2 GHz CPUs (4 cores each)
- 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)

Each CPU has 512K cache. The bus runs at 667 Mhz. The main memory is 192 GB Memory as 24 8GB modules. We anticipate that 16 processes can be run without time slicing.

The minimum instruction set across all processors used in the evaluation is specified here[7]. Dependence on instructions not included in the minimum instruction set is prohibited.

NIST is requiring use of 64 bit implementations throughout. This will support large memory allocation to support 1:N identification task with image and video frame counts in the millions. For still images, if all templates were to be held in memory, the 192GB capacity implies a limit of ~19KB per template, for a 10 million image enrollment. For video, given the data expectations and the occurrence of faces in the imagery, we anticipate the developers will have sufficient

[4] See the FRVT 2013 test report: NIST Interagency Report 8009, linked from http://face.nist.gov/frvt

[5] See the IREX III test report: NIST Interagency Report 7836, linked from http://iris.nist.gov/irex

[6] See the MBE-STILL 2010 test report, NIST Interagency Report 7709, linked from http://face.nist.gov/mbe

[7] cat /proc/cpuinfo returns fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm 3wext 3dnow constant_tsc nonstop_tsc pni cx16 popcnt lahf_lm cmp_legacy svm extapic cr8_legacy altmovcr8 abm sse4a misalignsse 3dnowprefetch osvw

memory for video templates. Note that while the API allows read access of the disk during the 1:N search, the disk is, of course, relatively slow.

Some of the section 3 API calls allow the implementation to write persistent data to hard disk. The amount of data shall not exceed 200 kilobytes per enrolled image. NIST will respond to prospective participants' questions on the hardware, by amending this section.

1.19. Operating system, compilation, and linking environment

The operating system that the submitted implementations shall run on will be released as a downloadable file accessible from http://nigos.nist.gov:8080/evaluations/, which is the 64-bit version of CentOS 7.0 running Linux kernel 3.10.0.

For this test, Windows machines will not be used. Windows-compiled libraries are not permitted. All software must run under Linux.

NIST will link the provided library file(s) to our C++ language test drivers. Participants are required to provide their library in a format that is linkable using g++ version 4.8.2. The standard libraries are:

/usr/lib64/libstdc++.so.6.0.19 lib64/libc.so.6 -> libc-2.17.so lib64/libm.so.6 -> libm-2.17.so

A typical link line might be

```
g++ -I. -Wall -m64 -o fivetest  fivetest.cpp  -L.  –lfive_Enron_A_07
```

The Standard C++ library should be used for development of the SDKs. The prototypes from the API of this document will be written to a file "five.h" which will be included via

```
#include <five.h>
```

The header files will be made available to implementers at http://nigos.nist.gov:8080/five.

NIST will handle all input of images via the JPEG and PNG libraries, sourced, respectively from http://www.ijg.org/ and see http://libpng.org.

All compilation and testing will be performed on x86 platforms. Thus, participants are strongly advised to verify library-level compatibility with g++ (on an equivalent platform) prior to submitting their software to NIST to avoid linkage problems later on (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.).

Dependencies on external dynamic/shared libraries such as compiler-specific development environment libraries are discouraged. If absolutely necessary, external libraries must be provided to NIST upon prior approval by the Test Liaison.

1.20. Software and documentation

1.20.1. SDK Library and platform requirements

Participants shall provide NIST with binary code only (i.e. no source code). Header files (".h") are allowed, but these shall not contain intellectual property of the company nor any material that is otherwise proprietary. It is preferred that the SDK be submitted in the form of a single static library file. However, dynamically linked shared library files are permitted.

The core library shall be named according to Table 5. Additional shared object library files may be submitted that support this "core" library file (i.e. the "core" library file may have dependencies implemented in these other libraries).

Intel Integrated Performance Primitives (IPP) libraries are permitted if they are delivered as a part of the developer-supplied library package. It is the provider's responsibility to establish proper licensing of all libraries. The use of IPP libraries shall not inhibit the SDK's ability to run on CPUs that do not support IPP. Please take note that some IPP functions are multithreaded and threaded implementations may complicate comparative timing.

Access to any GPUs is not permitted.

Table 5 – Implementation library filename convention

Form	libFIVE_provider_sequence.ending			
Underscore delimited parts of	libFIVE	provider	sequence	ending

the filename				
Description	First part of the name, required to be this.	Single word name of the main provider EXAMPLE: Acme	A two digit decimal identifier to start at 00 and increment by 1 every time any SDK is sent to NIST. EXAMPLE: 07	Either .so or .a
Example	libFIVE_Acme_C_07.a			

NIST will report the size of the supplied libraries.

1.20.2. Configuration and developer-defined data

The implementation under test may be supplied with configuration files and supporting data files. The total size of the SDK, that is all libraries, include files, data files and initialization files shall be less than or equal to 1 073 741 824 bytes = 1024^3 bytes.

NIST will report the size of the supplied configuration files.

1.20.3. Installation and Usage

The SDK must install easily (i.e. one installation step with no participant interaction required) to be tested, and shall be executable on any number of machines without requiring additional machine-specific license control procedures or activation.

The SDK shall be installable using simple file copy methods. It shall not require the use of a separate installation program.

The SDK shall neither implement nor enforce any usage controls or limits based on licenses, number of executions, presence of temporary files, etc. The submitted implementations shall remain operable with no expiration date.

Hardware (e.g. USB) activation dongles are not acceptable.

1.20.4. Hard disk space

FIVE participants should inform NIST if their implementations require more than 100K of persistent storage, per enrolled image on average.

1.20.5. Documentation

Participants shall provide complete documentation of the SDK and detail any additional functionality or behavior beyond that specified here. The documentation must define all (non-zero) developer-defined error or warning return codes.

1.20.6. Modes of operation

Individual SDKs provided shall not include multiple “modes” of operation, or algorithm variations. No switches or options will be tolerated within one library. For example, the use of two different “coders” by a feature extractor must be split across two separate SDK libraries, and two separate submissions.

1.21. Runtime behavior

1.21.1. Interactive behavior

The SDK will be tested in non-interactive “batch” mode (i.e. without terminal support). Thus, the submitted library shall not use any interactive functions such as graphical user interface (GUI) calls, or any other calls which require terminal interaction e.g. reads from “standard input”.

1.21.2. Error codes and status messages

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

1.21.3. Exception Handling

The application should include error/exception handling so that in the case of a fatal error, the return code is still provided to the calling application.

1.21.4. External communication

Processes running on NIST hosts shall not side-effect the runtime environment in any manner, except for memory allocation and release. Implementations shall not write any data to external resource (e.g. server, file, connection, or other process), nor read from such. If detected, NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

1.21.5. Stateless behavior

All components in this test shall be stateless, except as noted. This applies to face 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 will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

1.22. Threaded computations

Table 6 shows the limits on the numbers of threads a face recognition implementation may use. Threading is prohibited for feature extraction and search, because NIST will parallelize the test by dividing the workload across many cores and many machines. For the finalization function, if threading is used, NIST requires the provider to disclose the maximum number of threads to us.

Table 6 – Number of threads allowed for each function

Function	Video
Feature extraction	1
Finalize enrollment (before 1:N)	$1 \le T \le 16$
Identification	1

To expedite testing NIST will run up to P >> 1 processes concurrently. NIST's calling applications are single-threaded.

1.23. Time limits

The elemental functions of the implementations shall execute under the time constraints of Table 7. These time limits apply to the function call invocations defined in section 3. Assuming the times are random variables, NIST cannot regulate the maximum value, so the time limits are 90-th percentiles. This means that 90% of all operations should take less than the identified duration.

The time limits apply per image or video frame. When K images of a person are present or K frames are in a video clip, the time limits shall be increased by a factor K.

Table 7 – Processing time limits in milliseconds

Function	Video-to-Video	Video-to-Still	Still-to-Video
Feature extraction enrollment	5 * 1500 per video frame (1 core)	1500 per image (1 core)	5 * 1500 per video frame (1 core)
Feature extraction for identification	5 * 1500 per video frame (1 core)	1500 per image (1 core)	5 * 1500 per video frame (1 core)

For video: the multiple of K=5 is a notional average of the number of persons expected in any given frame. This figure is proportionally unreliable for any given sample.

While there is no time limit for the enrollment finalization procedure, NIST will report the execution duration.

1.24. Test datasets

This section is under development. The data has, in some cases, been estimated from initial small partitions. The completion of this section depends on further work. The information is subject to change. We intend to update this section as fully as possible.

NIST is likely to use other datasets, in addition.

Table 8 – Main video corpora (others will be used)

	Dataset P	Dataset T	Dataset B	Other datasets - Undisclosed
Collection, environment	Indoor public space with individuals walking mostly toward cameras as could occur on a transit terminal		Television footage, indoor and outdoor	
Number of individuals in field of view	Multiple, usually below 20 many not fully visible but usually more than 1.		Few, most often 1, occasionally others in background	
View angle	Various pitch due to different heights of camera installation, some yaw also due to subject behavior		Pitch variation present, but yaw angles vary more due to subject behavior	
Video frame size	1920 x 1080	Various	Various	
Eye to eye distance	10-100 pixels	10-150 pixels	10-120	
	The above values are guidelines; exceptions will inevitably occur in large datasets.			
Camera properties	Consumer-grade video	Professional-grade video	Professional-grade video cameras	
Camera motion	Fixed geometry, fixed optics		Usually camera is still or slowly panning or zooming	
Frames per second	24	Up to 30	Up to 30	
Similar composition to	Compare to the iLids data but with higher spatial resolution on the face.		Similar to YouTubeFaces in that typically one subject is present and in the foreground	
Accompanying stills	Yes, for video-to-still and still-to-video searches, high-resolution stills approximating ISO/IEC 19794-5 are available. In addition, off-angle images exist with many combinations of pitch and yaw. In addition, less formal "social-media" like stills are available also. Various galleries will be formed from these images. Images for which interocular distance exceeds 240 pixels will be downsized.		Stills usually resemble frames from the video. ISO/IEC 19794-5 images are not usually available.	

NIST does not know the minimum and maximum numbers of persons appearing in video sequences. Moreover, NIST will apply the algorithms to other databases. The maximum number of frames in a video sequence will be limited by the duration of the sequence. NIST expects to use sequences whose duration extends from a few seconds to a few minutes

Some notes regarding the video data:

- NIST does not anticipate using interlaced video.
- The videos are contiguous in time, without interruptions.
- Some sequences exist at much higher frame rates. NIST will examine whether this offers benefit.
- Some of the datasets were collected using consumer-grade cameras capturing video in standard formats while others were collected using professional-grade cameras captured in modern proprietary video codecs.

In some videos, the scenes capture people walking towards the camera. Occasionally, there are people walking in various transverse directions including people walking away from the camera. The cameras have varying pitch angles ranging from 0 degrees (frontal) to higher values. The depth of scene varies between the cameras such that the sizes of the faces vary, with the following:

— Eye-to-eye distances range from approximately 10 pixels to 120 pixels

— Amount of time a face is fully visible in a scene can vary from approximately 0 to 30 seconds

— Some of the captures include non-uniform lighting due to light coming through adjacent windows

Please note that the properties stated above may not hold for all datasets that might be employed in FIVE.

1.25. Ground truth integrity

Some of the test databases will be derived from operational systems. They may contain ground truth errors in which

— a single person is present under two different identifiers, or

— two persons are present under one identifier, or

— in which a face is not present in the image.

If these errors are detected, they will be removed. NIST will use aberrant scores (high impostor scores, low genuine scores) to detect such errors. This process will be imperfect, and residual errors are likely. For comparative testing, identical datasets will be used and the presence of errors should give an additive increment to all error rates. For very accurate implementations this will dominate the error rate. NIST intends to attach appropriate caveats to the accuracy results. For prediction of operational performance, the presence of errors gives incorrect estimates of performance.

2. Data structures supporting the API

2.1. Overview

This section describes the API for the face recognition in video applications described in section 1.8. All SDK's submitted to FIVE shall implement the functions required in Section 3.

2.2. Requirement

FIVE participants shall submit an SDK which implements the relevant C++ prototyped interfaces of clause 3. C++ was chosen in order to make use of some object-oriented features.

2.3. File formats and data structures

2.3.1. Overview

In this test, an individual is represented by K ≥ 1 two-dimensional facial images, and by subject and image-specific metadata.

2.3.2. Dictionary of terms describing images and MULTIFACEs

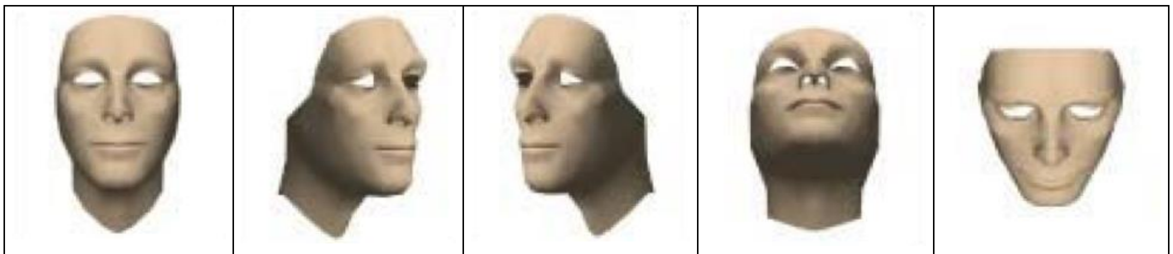
Images will be accompanied by one of the labels given in Table 9. Face recognition implementations submitted to FIVE should tolerate images of any category.

Table 9 – Labels describing types of images

	Label as C++ string	Meaning	Yaw (degrees)	Pitch (degrees)
1.	"unknown"	Either the label is unknown or unassigned.		
2.	"uncontrolled"	Any illumination, pose is unknown and could be frontal		
3.	"FF"	Full frontal	0	0
4.	"FD"	Face down	0	10 to 40

5.	“FU”	Face up	0	-10 to -40
6.	“QL”	Quarter left	-10 to -45	0
7	“QR”	Quarter right	10 to 45	0
8.	“HL”	Half left	-46 to -80	0
9.	“HR”	Half right	46 to 80	0
10.	“PL”	Profile left	-90	0
11.	“PR”	Profile right	90	0
12.	“QLU”	Quarter left up	-10 to -45	-10 to -40
13.	“QRU”	Quarter right up	10 to 45	-10 to -40
14.	“HLU”	Half left up	-46 to -80	-10 to -40
15.	“HRU”	Half right up	46 to 80	-10 to -40
16.	“HLD”	Half left down	-46 to -80	10 to 40
17.	“HRD”	Half right down	46 to 80	10 to 40

Figure 2 provides examples of pose angles and their encoding (yaw, pitch) as specified in the ISO/IEC 19794-5 [ISO], with yaw angle defined as the rotation in degrees about the y-axis (vertical axis) and pitch angle defined as the rotation in degrees about the x-axis (horizontal axis).



(0,0)	(+45,0)	(-45,0)	(0,-45)	(0,+45)

Figure 2 – Examples of pose angles and their encodings (yaw, pitch)

NOTE 1: We do not intend to deliberately include non-face images in this test.

NOTE 2: **MULTIFACE**s will contain face images of only one person.

A **MULTIFACE** (see Table 12) will be accompanied by one of the labels given in Table 10. Face recognition implementations submitted to FIVE should tolerate **MULTIFACE**s of any category.

Table 10 – Labels describing types of MULTIFACEs

	Label as C++ string	Meaning
1.	“FRONTAL”	All **ONEFACE**s contain nominally frontal images and are labeled “FF”.
2.	“MULTIPOSE”	Each **ONEFACE** is labeled with one of the following: “FF”, “FD”, “FU”, “QL”, “QR”, “HL”, “HR”, “PL”, “PR”, “QLU”, “QRU”, “HLU”, “HRU”, “HLD”,“HRD”.
3.	“INFORMAL”	All **ONEFACE**s contain informal images that are labeled “uncontrolled”.
4.	“UNKNOWN”	Each **ONEFACE** is labeled with one of the labels from Table 9, including possibly “unknown” or “uncontrolled”.

2.3.3. Data structures for encapsulating multiple still images

The standardized formats for facial images are the ISO/IEC 19794-5:2005 and the ANSI/NIST ITL 1-2007 type 10 record. The ISO record can store multiple images of an individual in a standalone binary file. In the ANSI/NIST realm, K images of an individual are usually represented as the concatenation of one Type 1 record + K Type 10 records. The result is usually stored as an EFT file.

An alternative method of representing K images of an individual is to define a structure containing an image filename and metadata fields. Each file contains a standardized image format, e.g. PNG (lossless) or JPEG (lossy).

2.3.4. Class for encapsulating a single face image

Table 11 – ONEFACE class

	C++ code fragment	Remarks
1.	`class ONEFACE`	
2.	`{` `private:`	
3.	`uint16_t imageWidth;`	Number of pixels horizontally
4.	`uint16_t imageHeight;`	Number of pixels vertically
5.	`uint16_t imageDepth;`	Number of bits per pixel. Legal values are 8 and 24.
6.	`uint8_t format;`	Flag indicating native format of the image as supplied to NIST 0x01 = JPEG (i.e. compressed data) 0x02 = PNG (i.e. never compressed data)
7.	`uint8_t *data;`	Pointer to raster scanned data. Either RGB color or intensity. If image_depth == 24 this points to 3WH bytes RGBRGBRGB... If image_depth == 8 this points to WH bytes `IIIIIII`
8.	`std::string description;`	Single description of the image. The allowed values for this string are given in Table 9.
9.	`public:` `//getter/setter methods`	
10.	`};`	

2.3.5. Class for encapsulating a set of face images from a single person

Table 12 – MULTIFACE class

	C++ code fragment	Remarks

1.	`class MULTIFACE`
	`{`
	`private:`
	`std::vector<ONEFACE> faces;`
		Vector containing F pre-allocated face images of the same person. The number of items stored in the vector is accessible via the vector::size() function.

2.	`std::string description;`	Single description of the vector of ONEFACEs. The allowed values for this string are given in Table 10.
3.	`public:` `//getter/setter methods`	
4.	`};`	

2.3.6. Dictionary of terms describing ONEVIDEOs

A **ONEVIDEO** will be accompanied by one of the labels given in Table 13, describing the density of people in the video frames. Face recognition implementations submitted to FIVE should tolerate ONEVIDEOs of any category.

Table 13 – Labels describing the density of people in the video frames

	Label as C++ string	Meaning
1.	"SINGLE"	All of the video frames contain one and only one person. Such video might arise from a TV interview or speech. An algorithm should produce one template from the ONEVIDEO.
2.	"UNKNOWN"	Video frames can contain zero or more people in each frame. Such video might arise in a surveillance clip. The number of templates to return would be a random variable.

2.3.7. Class for encapsulating a video sequence

Table 14 – ONEVIDEO Class

	C++ code fragment	Remarks

1.	class **ONEVIDEO**	
2.	{ private:	
3.	uint16_t frameWidth;	Number of pixels horizontally of all frames
4.	uint16_t frameHeight;	Number of pixels vertically of all frames
5.	uint8_t frameDepth;	Number of bits per pixel for all frames. Legal values are 8 and 24.
6.	uint16_t framesPerSec;	The frame rate of the video sequence. If this value is 0, the frames are sampled irregularly and perhaps infrequently from the parent video clip (e.g. manually selected frames, or just the I-frames).
7.	std::string peopleDensity;	Single description of the density of people in the video frames. The allowed values for this string are given in Table 13.
8.	std::vector<const uint8_t*> data;	Vector of pointers to data from each frame in the video sequence. The number of frames (i.e. size of the vector) can be obtained by calling vector::size(). The i-th entry in data (ie. data[i]) points to frame_width x frame_height pixels of data for the i-th frame.
9.	public:	
10.	//getter/setter methods	
11.	};	

2.3.8. Class representing a pair of eye coordinates

The data structure for reporting person locations in video appears in Table 15. The coordinates may be useful to NIST for relating spatial location to recognition success during our analysis.

Table 15 – EYEPAIR Class

	C++ code fragment	Remarks
1.	class **EYEPAIR**	
2.	{ private:	
3.	bool isSet;	If the eye coordinates have been computed and assigned successfully, this value should be set to true, otherwise it should be set to false.
4.	int16_t xLeft; int16_t yLeft;	X and Y coordinate of the center of the subject's left eye. Out-of-range values (e.g. x < 0 or x >= width) indicate the implementation believes the eye center is outside the image.
5.	int16_t xRight; int16_t yRight;	X and Y coordinate of the center of the subject's right eye. Out-of-range values (e.g. x < 0 or x >= width) indicate the implementation believes the eye center is outside the image.
6.	uint16_t frameNum	For video: the frame number that corresponds to the video frame from which the eye coordinates were generated. (ie. the i-th frame from the video sequence). This field should not be set for eye coordinates for a single still image.
7.	public: //getter/setter methods	
8.	};	

2.3.9. Data type for representing a person's trajectory via eye coordinates from a video sequence

Table 16 – PersonTrajectory typedef

	C++ code fragment	Remarks
1.	typedef std::vector<**EYEPAIR**> PersonTrajectory;	Vector of **EYEPAIR** (see 2.3.8) objects for video frames where eyes were detected. This data structure should store eye coordinates for each video frame where eyes were detected for a particular person. For video frames where the person's eyes were not detected, the SDK shall not add an **EYEPAIR** to this data structure. If a face can be detected, but not the eyes, the implementation should nevertheless fill this data structure with $(x,y)_{LEFT} == (x,y)_{RIGHT}$ representing some point on the center of the face.

2.3.10. Class for representing a person from a video sequence or an image

Table 17 – PERSONREP Class

	C++ code fragment	Remarks
1.	`class PERSONREP`	
2.	`{` `private:`	
3.	`PersonTrajectory eyeCoordinates;`	Data structure for capturing eye coordinates
4.	`PersonTemplate proprietaryTemplate;`	PersonTemplate is a wrapper to a uint8_t* for capturing proprietary template data representing a person from a video sequence or an image.
5.	`public:`	
6.	`PERSONREP(const uint64_t inSize);`	The constructor takes a size parameter and allocates memory of *inSize*. getPersonTemplatePtr() should be called to access the newly allocated memory for SDK manipulation. Please note that this class will take care of all memory allocation and de-allocation of its own memory. The SDK shall not de-allocate memory created by this class.
7.	`void pushBackEyeCoord(const EYEPAIR &eyes);`	This function should be used to add **EYEPAIR**s for the video frames or images where eye coordinates were detected.
8.	`uint8_t* getPersonTemplatePtr();`	This function returns a uint8_t* to the template data.
9.	`uint64_t getPersonTemplateSize() const;`	This function returns the size of the template data.
10.	`//… getter methods, copy constructor,` `//… assignment operator`	
11.	`};`	

2.3.11. Class for result of an identification search

All identification searches shall return a candidate list of a NIST-specified length. The list shall be sorted with the most similar matching entries list first with lowest rank.

Table 18 – CANDIDATE Class

	C++ code fragment	Remarks
1.	`class CANDIDATE`	
2.	`{` `private:`	
3.	`bool isSet`	If the candidate is valid, this should be set to true. If the candidate computation failed, this should be set to false.
4.	`uint32_t templateId;`	The Template ID integer from the enrollment database manifest defined in clause 2.3.6.
5.	`double similarityScore;`	Measure of similarity between the identification template and the enrolled candidate. Higher scores mean more likelihood that the samples are of the same person. An algorithm is free to assign any value to a candidate. The distribution of values will have an impact on the appearance of a plot of false-negative and false-positive identification rates.
6.	`public:` `//getter/setter methods`	
7.	`};`	

2.3.12. Data type for representing a list of results of an identification search

Table 19 – CANDIDATELIST typedef

	C++ code fragment	Remarks
1.	`typedef std::vector<CANDIDATE> CANDIDATELIST;`	A vector containing objects of **CANDIDATE**s. The **CANDIDATE** class is defined in section 2.3.11.

2.3.13. Class representing return code values

Table 20 – ReturnCode class

	C++ code fragment	Remarks
	`class ReturnCode {` `public:`	
1.	`typedef enum`	
2.	`{`	
3.	`Success=0,`	Success
4.	`MissingConfig=1,`	The configuration data is missing or unreadable
5.	`EnrollDirFailed=2,`	An operation on the enrollment directory failed
6.	`InitNumData=3,`	The SDK can't support the number of images or videos
7.	`InitBadDesc=4,`	The image descriptions are unexpected or unusable
8.	`RefuseInput=5,`	Elective refusal to process this kind of input (**ONEVIDEO** or MULTIFACE)
9.	`FailExtract=6,`	Involuntary failure to extract features
10.	`FailTempl=7,`	Elective refusal to produce a template
11.	`FailParse=8,`	Cannot parse input data
12.	`FinInputData=9,`	Cannot locate input data
13.	`FinTemplFormat=10,`	One or more template files are in an incorrect format
14.	`IdBadTempl=11,`	The input template was defective
15.	`ImgSizeNotSupported=12,`	Size of input image/frame not supported
16.	`Vendor=13`	Vendor-defined failure
17.	`} Status;`	
18.	`ReturnCode(const Status inStatus);`	Constructor that takes an input parameter of a Status enum value. All of the functions that need to be implemented for the Video API return an instantiation of a ReturnCode object with a valid status value passed in as a parameter.
19.	`Status getStatus() const;`	Getter method to return status value
20.	`private:`	
21.	`Status status;`	Member variable for storing status
22.	`};`	

2.4. File structures for enrolled template collection

For still image enrollment, an SDK converts a **MULTIFACE** into a template using the ImageEnrollment::generateEnrollmentTemplate() function of section 3.3.5.2. For video enrollment, an SDK converts a **ONEVIDEO** into one or more templates, using the VideoEnrollment::generateEnrollmentTemplate() of section 3.3.1.2. To support the identification functions, NIST will concatenate enrollment templates into a single large file. This file is called the EDB (for enrollment database). The EDB is a simple binary concatenation of proprietary templates. There is no header. There are no delimiters. The EDB may extend to hundreds of gigabytes in length.

This file will be accompanied by a manifest; this is an ASCII text file documenting the contents of the EDB. The manifest has the format shown as an example in Table 21. If the EDB contains N templates, the manifest will contain N lines. The fields are space (ASCII decimal 32) delimited. There are three fields, all containing numeric integers. Strictly speaking, the third column is redundant.

Table 21 – Enrollment dataset template manifest

Field name	Template ID	Template Length	Position of first byte in EDB
Datatype required	Unsigned decimal integer	Unsigned decimal integer	Unsigned decimal integer
Datatype length required	4 bytes	4 bytes	8 bytes
Example lines of a manifest file appear to the right. Lines 1, 2, 3 and N appear.	90201744	1024	0
	163232021	1536	1024
	7456433	512	2560
	…		

	183838	1024	307200000

The EDB scheme avoids the file system overhead associated with storing millions of individual files.

3. API Specification

3.1.1. Definitions

As shown in Table 22, the video API supports 1:N identification of video-to-video, video-to-still image, and still image-to-video. The following hold:

- A still image is a picture of one and only one person. One or more such images are presented to the implementation using a **MULTIFACE** data structure.
- A video is a sequence of $F \geq 1$ frames containing $P \geq 0$ persons.
- A frame is 2D still image containing $P \geq 0$ persons.
- Any person might be present in $0 \leq f \leq F$ frames, and their presence may be non-contiguous (e.g. due to occlusion).
- Different videos contain different numbers of frames and people.
- A **ONEVIDEO** container is used to represent a video. It contains a small header and pointers to F frames.
- Any person found in a video is represented by proprietary template (feature) data contained with a **PERSONREP** data structure. A proprietary template contains information from one or more frames. Internally, it might embed multiple traditional still-image templates, or it might integrate feature data by tracking a person across multiple frames.
- A **PERSONREP** structure additionally contains a trajectory indicating the location of the person in each frame.

All of the code for the classes needed to implement the video API will be provided to implementers at http://nigos.nist.gov:8080/five. A single sample video has been made available at the same link. The sample video is only approximately representative of the scene and is not an extraction from the actual video data that will be used in the evaluation. It is only intended to illustrate similarities in terms of camera placement relative to the subject and people behavior. It is not intended to represent the optical properties of the actual imaging systems, particularly the spatial sampling rate, nor the compression characteristics.

Table 22 – API implementation requirements for FIVE

Function	Video-to-video	Still-to-video	Video-to-still
Enroll	Videos	Videos	Stills
Enrollment input datatype	**ONEVIDEO**	**ONEVIDEO**	**MULTIFACE**
Enrollment datatype	**PERSONREP**	**PERSONREP**	**PERSONREP**
Search	Video	Still	Video
Search input datatype	**ONEVIDEO**	**MULTIFACE**	**ONEVIDEO**
Search datatype	**PERSONREP**	**PERSONREP**	**PERSONREP**
Search result	**CANDIDATELIST**	**CANDIDATELIST**	**CANDIDATELIST**
API requirements	3.3.1 + 3.3.2 + 3.3.3 + 3.3.4	3.3.1 + 3.3.2 + 3.3.7 + 3.3.4	3.3.5 + 3.3.6 + 3.3.3 + 3.3.8

3.1.1.1. Video-to-video

Video-to-video identification is the process of enrolling N videos and then searching the enrollment database with a search video. During identification, the SDK shall return a set of indices of candidate videos that contain people who appear in the search video.

- N templates will be generated from M enrollment videos. If no people appear in the videos, N will be 0. If many people appear in each video, we'd expect $N > M$.
- The N templates will be concatenated and finalized into a proprietary enrollment data structure.
- A **ONEVIDEO** will be converted to $S \geq 0$ identification template(s) based on the number of people detected in the video.

- Each identification template generated will be searched against the enrollment database of templates generated from the M input videos.
- We anticipate that the same person may appear in more than one enrolled video.

3.1.1.2. Still image-to-video

Still image-to-video identification is the process of enrolling N videos and then searching the enrollment database with a template produced from a **MULTIFACE** as follows:

- N templates will be generated from 1 < M ≤ N enrollment videos.
- The N templates will be concatenated and finalized into a proprietary enrollment data structure.
- A **MULTIFACE** (still image) will be converted to an identification template.
- The identification template will be searched against the enrollment database of N templates.
- We anticipate that the same person may appear in more than one enrolled video.

3.1.1.3. Video-to-still image

Video-to-still image identification is the process of enrolling N **MULTIFACEs** (see Table 12) and then searching the enrollment database with templates from persons found in a video as follows

- N templates will be generated from N still-image **MULTIFACE**s.
- The N templates will be concatenated and finalized into a proprietary enrollment data structure.
- A **ONEVIDEO** will be converted to S ≥ 0 identification template(s) based on the number of people detected in the video.
- Each of the S identification templates will be searched separately against the enrollment database of N templates.

3.2. 1:N Identification

3.2.1. Overview

The 1:N application proceeds in two phases, enrollment and identification. The identification phase includes separate pre-search feature extraction stage, and a search stage.

The design reflects the following *testing* objectives for 1:N implementations.

- support distributed enrollment on multiple machines, with multiple processes running in parallel
- allow recovery after a fatal exception, and measure the number of occurrences
- allow NIST to copy enrollment data onto many machines to support parallel testing
- respect the black-box nature of biometric templates
- extend complete freedom to the provider to use arbitrary algorithms
- support measurement of duration of core function calls
- support measurement of template size

Table 23 – Procedural overview of the identification test

Phase	#	Name	Description	Performance Metrics to be reported by NIST

Enrollment	E1	Initialization	For still image enrollment, give the implementation advance notice of the number of individuals and images that will be enrolled. Give the implementation the name of a directory where any provider-supplied configuration data will have been placed by NIST. This location will otherwise be empty. The implementation is permitted **read-write-delete access** to the enrollment directory during this phase. The implementation is permitted read-only access to the configuration directory. After enrollment, NIST may rename and relocate the enrollment directory - the implementation should not depend on the name of the enrollment directory.	
	E2	Parallel Enrollment	For still image enrollment, for each of N individuals, pass multiple images to the implementation for conversion to a combined template. For video enrollment, for each of M video clips, pass multiple video frames to the implementation for generation of N templates, based on the number of people detected in the videos. The implementation will return a template to the calling application. The implementation is permitted **read-only access** to the enrollment directory during this phase. NIST's calling application will be responsible for storing all templates as binary files. These will not be available to the implementation during this enrollment phase. Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers. For still image enrollment, the same person will not be enrolled twice.	Statistics of the times needed to enroll an individual or video clip. Statistics of the sizes of created templates. The incidence of failed template creations.
	E3	Finalization	Permanently finalize the enrollment directory. This supports, for example, dis-interleaving of internal feature representations, writing of a manifest, indexing, tree construction, computation of statistical information over the enrollment dataset, and adaptation of the representation. The implementation is permitted **read-write-delete access** to the enrollment directory during this phase.	For still image enrollment, size of the enrollment database as a function of population size N and the number of images. Duration of this operation. The time needed to execute this function shall be reported with the preceding enrollment times.
Pre-search	S1	Initialization	Tell the implementation the location of an enrollment directory. The implementation could look at the enrollment data. The implementation is permitted **read-only access** to the enrollment directory during this phase.	Statistics of the time needed for this operation.
	S2	Template preparation	For each probe, create a template from a set of input images or one or more templates from a set of video clips. This operation will generally be conducted in a separate process invocation to step S2. The implementation is **permitted no access** to the enrollment directory during this phase. The result of this step is a search template.	Statistics of the time needed for this operation. Statistics of the size of the search template(s).
Search	S3	Initialization	Tell the implementation the location of an enrollment directory. The implementation should read all or some of the enrolled data into main memory, so that searches can commence. The implementation is permitted **read-only access** to the enrollment directory during this phase.	Statistics of the time needed for this operation.
	S4	Search	A template or multiple templates is searched against the enrollment database. The implementation is permitted **read-only access** to the enrollment directory during this phase.	Statistics of the time needed for this operation. Accuracy metrics - Type I + II error rates.

				Failure rates.

3.3. Interfaces

3.3.1. The VideoEnrollment Interface

The abstract class VideoEnrollment must be implemented by the SDK developer in a class named exactly SdkVideoEnrollment. The processing that takes place during each phase of the test is done via calls to the methods declared in the interface as pure virtual, and therefore is to be implemented by the SDK. The test driver will call these methods, handling all return values.

	C++ code fragment	Remarks
1.	`class VideoEnrollment`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir,`
	`const uint32_t numVideos) = 0 ;`
		Initialize the enrollment session.

4.	`virtual ReturnCode generateEnrollmentTemplate(`
	`const` **`ONEVIDEO`** `&inputVideo,`
	`vector<`**`PERSONREP`**`> &enrollTemplates) = 0;`
		Generate enrollment template(s) for the persons detected in the input video. This function takes a **ONEVIDEO** (see 2.3.6) as input and populates a vector of **PERSONREP** (see 2.3.10) with the number of persons detected from the video sequence. The implementation could call vector::push_back to insert into the vector.

5.	`// Destructor`	
6.	`};`	

3.3.1.1. Initialization of the video enrollment session

Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization below for video-to-video and still image-to-video.

Table 24 – VideoEnrollment::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir,	Input
	const uint32_t numVideos);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called N=1 times by the NIST application immediately before any M ≥ 1 calls to generateEnrollmentTemplate. Caution: The implementation should tolerate execution of P > 1 processes on the one or more machines each of which may be reading and writing to this same enrollment directory in parallel. File locking or process-specific temporary filenames would be needed to safely write content in the enrollDir.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. When this function is called, the SDK may populate this folder in any manner it sees fit. Permissions will be read-write-delete.
	numVideos	The total number of videos that will be passed to the SDK for enrollment.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	InitNumData	The SDK cannot support the number of videos.
	Vendor	Vendor-defined failure

3.3.1.2. Video enrollment

A **ONEVIDEO** is converted to enrollment template(s) for each person detected in the **ONEVIDEO** using the function below.

Table 25 – VideoEnrollment::generateEnrollmentTemplate

Prototypes	ReturnCode generateEnrollmentTemplate(	
	const **ONEVIDEO** &inputVideo,	Input
	std::vector<**PERSONREP**> &enrollTemplates);	Output
Description	This function takes a **ONEVIDEO**, and outputs a vector of **PERSONREP** objects. If the function executes correctly (i.e. returns a ReturnCode::Success exit status), the NIST calling application will store the template. The NIST application will concatenate the templates and pass the result to the enrollment finalization function. For a video in which no persons appear, a valid output is an empty vector (i.e. size() == 0). If the function gives a non-zero exit status: – If the exit status is ReturnCode::FailParse, NIST will debug, otherwise – the test driver will ignore the output template (the template may have any size including zero) – the event will be counted as a failure to enroll. Such an event means that this person can never be identified correctly. IMPORTANT. NIST's application writes the template to disk. The implementation must not attempt writes to the enrollment directory (nor to other resources). Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function.	
Input Parameters	inputVideo	An instance of a Table 14 class.
Output Parameters	enrollTemplates	For each person detected in the **ONEVIDEO**, the function shall identify the person's estimated eye centers for each video frame where the person's eye coordinates can be calculated. The eye coordinates shall be captured in the **PERSONREP**.eyeCoordinates variable, which is a vector of **EYEPAIR** objects. The frame number from the video of where the eye coordinates were detected shall be captured in the **EYEPAIR**.frameNum variable for each pair of eye coordinates. In the event the eye centers cannot be calculated (ie. the person becomes out of sight for a few frames in the video), the SDK shall not store an **EYEPAIR** for those frames.
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of **ONEVIDEO**
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	ImgSizeNotSupported	Input image/frame size too small or large
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.3.2. The VideoFinalize Interface

The abstract class VideoFinalize must be implemented by the SDK developer in a class named exactly SdkVideoFinalize. The finalize function in this class takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 2.3.6. The enrollment directory permissions will be read + write.

	C++ code fragment	Remarks
1.	`class VideoFinalize`	
2.	`{` `public:`	

3.	`virtual ReturnCode finalize(`
	`const string &enrollDir,`
	`const string &edbName,`
	`const string &edbManifest) = 0;`
		This function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete.

4.	`// Destructor`	

5.	};	

3.3.2.1. Finalize video enrollment

After all templates have been created, the function of Table 26 will be called. This freezes the enrollment data. After this call the enrollment dataset will be forever read-only. This API does not support interleaved enrollment and search phases.

The function allows the implementation to conduct, for example, statistical processing of the feature data, indexing and data re-organization. The function may alter the file structure. It may increase or decrease the size of the stored data. No output is expected from this function, except a return code.

Table 26 – VideoFinalize::finalize

Prototypes	ReturnCode finalize (	
	const string &enrollDir,	Input
	const string &edbName,	Input
	const string &edbManifest);	Input
Description	This function takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 2.3.6. The enrollment directory permissions will be read + write. The function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete. This function should be tolerant of being called two or more times. Second and third invocations should probably do nothing.	
Input Parameters	enrollDir	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
	edbName	The name of a single file containing concatenated templates, i.e. the EDB of section 2.3.6. While the file will have read-write-delete permission, the SDK should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name.
	edbManifest	The name of a single file containing the EDB manifest of section 2.3.6. The file may be opened directly. It is not necessary to prepend a directory name.
Output Parameters	None	
ReturnCode	Success	Success
	FinInputData	Cannot locate the input data - the input files or names seem incorrect.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	FinTemplFormat	One or more template files are in an incorrect format.
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.3.3. The VideoFeatureExtraction Interface

The abstract class VideoFeatureExtraction must be implemented by the SDK developer in a class named exactly SdkVideoFeatureExtraction.

	C++ code fragment	Remarks
1.	`class VideoFeatureExtraction`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir) = 0;`
		Initialize the feature extraction session.

4.	`virtual ReturnCode generateIdTemplate(`
	`    const ONEVIDEO &inputVideo,`
	`    vector<PERSONREP> &idTemplates) = 0;`
		Generate identification template(s) for the persons detected in the input video. This function takes a **ONEVIDEO** (see 2.3.6) as input and populates a vector of **PERSONREP** (see 2.3.10) with the number of persons detected from the video sequence. The implementation could call vector::push_back to insert into the vector.

5.	`// Destructor`	
6.	`};`	

3.3.3.1. Video feature extraction initialization

Before one or more **ONEVIDEO**s are sent to the identification feature extraction function, the test harness will call the initialization function below.

Table 27 – VideoFeatureExtraction::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called once by the NIST application immediately before any M ≥ 1 calls to generateIdTemplate. The implementation has read-only access to enrollDir (containing prior enrollment data) and to configDir.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The read-only top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

3.3.3.2. Video feature extraction

A **ONEVIDEO** is converted to one or more identification templates using the function below. The result may be stored by NIST, or used immediately. The SDK shall not attempt to store any data.

Table 28 – VideoFeatureExtraction::generateIdTemplate

Prototypes	ReturnCode generateIdTemplate(	
	const **ONEVIDEO** &inputVideo,	Input
	std::vector<**PERSONREP**> &idTemplates);	Output
Description	This function takes a **ONEVIDEO** (see 2.3.6) as input and populates a vector of **PERSONREP** (see 2.3.10) with the number of persons detected from the video sequence. The implementation could call vector::push_back to insert into the vector. If the function executes correctly, it returns a zero exit status. The NIST calling application may commit the template to permanent storage, or may keep it only in memory (the implementation does not need to know). If the function returns a non-zero exit status, the output template will be not be used in subsequent search operations. The function shall not have access to the enrollment data, nor shall it attempt access.	
Input Parameters	InputVideo	An instance of a section 2.3.6 class. Implementations must alter their behavior according to the people detected in the video sequence.
Output Parameters	IdTemplates	For each person detected in the video, the function shall create a **PERSONREP** (see section 2.3.10) object, populate it with a template and eye coordinates for each frame where eyes

		were detected, and add it to the vector.
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of **ONEVIDEO**
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	ImgSizeNotSupported	Input image/frame size too small or large
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.3.4. The VideoSearch Interface

The abstract class VideoSearch must be implemented by the SDK developer in a class named exactly SdkVideoSearch.

	C++ code fragment	Remarks
1.	`class VideoSearch`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir) = 0;`
		Initialize the search session.

4.	`virtual ReturnCode identifyVideo(`
	`const PERSONREP &idVideoTemplate,`
	`const uint32_t candListLength,`
	`CANDIDATELIST &candList) = 0;`
		For video-to-video identification
		This function searches a template generated from a **ONEVIDEO** against the enrollment set, and outputs a vector containing candListLength objects of Candidates (see section 2.3.12).

5.	`virtual ReturnCode identifyImage(`
	`const PERSONREP &idImageTemplate,`
	`const uint32_t candListLength,`
	`CANDIDATELIST &candList) = 0;`
		For still-to-video identification
		This function searches a template generated from a **MULTIFACE** against the enrollment set, and outputs a vector containing candListLength objects of Candidates.

6.	`// Destructor`	
7.	`};`	

3.3.4.1. Video identification initialization

The function below will be called once prior to one or more calls of the searching function of Table 30. The function might set static internal variables so that the enrollment database is available to the subsequent identification searches.

Table 29 – VideoSearch::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir);	Input
Description	This function reads whatever content is present in the enrollment_directory, for example a manifest placed there by the VideoFinalize::finalize function.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The read-only top-level directory in which enrollment data was placed.
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

3.3.4.2. Video identification search

The function below compares a proprietary identification template against the enrollment data and returns a candidate list.

Table 30 – VideoSearch::identifyVideo and VideoSearch::identifyImage

Prototype	ReturnCode identifyVideo(		Searches a template generated from a **ONEVIDEO** against the enrollment set (video-to-video)
	const **PERSONREP** &idVideoTemplate,		Input
	const uint32_t candListLength,		Input
	CANDIDATELIST &candList);		Output
	ReturnCode identifyImage(		Searches a template generated from a **MULTIFACE** against the enrollment set (still-to-video)
	const **PERSONREP** &idImageTemplate,		Input
	const uint32_t candListLength,		Input
	CANDIDATELIST &candList);		Output
Description	This function searches an identification template against the enrollment set, and outputs a vector containing candListLength Candidates (see section 2.3.12). Each candidate shall be populated by the implementation and added to candList. Note that candList will be an empty vector when passed into this function. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.		
Input Parameters	idTemplate	A template from generateIdTemplate() - If the value returned by that function was non-zero the contents of idTemplate will not be used and this function (i.e. identifyVideo) will not be called.	
	candListLength	The number of candidates the search should return	
Output Parameters	candList	A vector containing candListLength objects of Candidates. The datatype is defined in section 2.3.12. Each candidate shall be populated by the implementation and added to this vector. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.	
ReturnCode	Success	Success	
	IdBadTempl	The input template was defective.	
	Vendor	Vendor-defined failure	

3.3.5. The ImageEnrollment Interface

The abstract class ImageEnrollment must be implemented by the SDK developer in a class named exactly SdkImageEnrollment.

	C++ code fragment	Remarks
1.	`class ImageEnrollment`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir,`
	`const uint32_t numPersons,`
	`const uint32_t numImages,`
	`const vector<string> &descriptions) = 0 ;`
		Initialize the enrollment session.

4.	`virtual ReturnCode generateEnrollmentTemplate(`
	`const MULTIFACE &inputFaces,`
	`PERSONREP &outputTemplate) = 0;`
		This function takes a **MULTIFACE** (see 2.3.3) as input and outputs a proprietary template represented by a **PERSONREP** (see 2.3.10).
		
		For each input image in the **MULTIFACE**, the function shall return the estimated eye centers by setting **PERSONREP**.eyeCoordinates.

5.	`// Destructor`	
6.	`};`	

3.3.5.1. Initialization of the image enrollment session

Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization below for video-to-still.

Table 31 – ImageEnrollment::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir,	Input
	const uint32_t numPersons,	Input
	const uint32_t numImages,	Input
	const std::vector<string> &descriptions);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called N=1 times by the NIST application immediately before any M ≥ 1 calls to generateEnrollmentTemplate. Caution: The implementation should tolerate execution of P > 1 processes on the one or more machines each of which may be reading and writing to this same enrollment directory in parallel. File locking or process-specific temporary filenames would be needed to safely write content in the enrollDir.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. When this function is called, the SDK may populate this folder in any manner it sees fit. Permissions will be read-write-delete.
	numPersons	The number of persons who will be enrolled.
	numImages	The total number of images that will be enrolled, summed over all identities.
	descriptions	A lexicon of labels one of which will be assigned to each enrollment image. See Table 9 for valid values. NOTE: The identification search images may or may not be labeled. An identification image may carry a label not in this set of labels. The number of items stored in the vector is accessible via the vector::size() function.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	InitNumData	The SDK cannot support the number of videos.
	InitBadDesc	The descriptions are unexpected, or unusable.
	Vendor	Vendor-defined failure

3.3.5.2. Image enrollment

A **MULTIFACE** (see Table 12) is converted to a single enrollment template using the function below.

Table 32 – ImageEnrollment::generateEnrollmentTemplate

Prototypes	ReturnCode generateEnrollmentTemplate(	
	const **MULTIFACE** &inputFaces,	Input
	PERSONREP &outputTemplate);	Output
Description	This function takes a **MULTIFACE**, and outputs a proprietary template in the form of a **PERSONREP** object. If the function executes correctly (i.e. returns a ReturnCode::Success exit status), the NIST calling application will store the template. The NIST application will concatenate the templates and pass the result to the enrollment finalization function. If the function gives a non-zero exit status: – If the exit status is ReturnCode::FailParse, NIST will debug, otherwise – the test driver will ignore the output template (the template may have any size including zero) – the event will be counted as a failure to enroll. Such an event means that this person can never be identified	

	correctly. IMPORTANT. NIST's application writes the template to disk. The implementation must not attempt writes to the enrollment directory (nor to other resources). Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function.	
Input Parameters	inputFaces	An instance of a **Table 12** structure.
Output Parameters	outputTemplate	An instance of a section 2.3.10 class, which stores proprietary template data and eye coordinates. The function shall identify the person's estimated eye centers for each image in the **MULTIFACE**. The eye coordinates shall be captured in the **PERSONREP**.eyeCoordinates variable, which is a vector of **EYEPAIR** objects. In the event the eye centers cannot be calculated, the SDK shall store an **EYEPAIR** and set **EYEPAIR**.isSet to false to indicate there was a failure in generating eye coordinates. In other words, for N images in the **MULTIFACE**.
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of **MULTIFACE**
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	ImgSizeNotSupported	Input image/frame size too small or large
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.3.6. The ImageFinalize Interface

The abstract class ImageFinalize must be implemented by the SDK developer in a class named exactly SdkImageFinalize. The finalize function in this class takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 2.3.6. The enrollment directory permissions will be read + write.

	C++ code fragment	Remarks
1.	`class ImageFinalize`	
2.	`{` `public:`	

3.	`virtual ReturnCode finalize(`
	`const string &enrollDir,`
	`const string &edbName,`
	`const string &edbManifest) = 0;`
		This function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete.

4.	`// Destructor`	
5.	`};`	

3.3.6.1. Finalize image enrollment

After all templates have been created, the function of Table 33 will be called. This freezes the enrollment data. After this call the enrollment dataset will be forever read-only. This API does not support interleaved enrollment and search phases.

The function allows the implementation to conduct, for example, statistical processing of the feature data, indexing and data re-organization. The function may alter the file structure. It may increase or decrease the size of the stored data. No output is expected from this function, except a return code.

Table 33 – ImageFinalize::finalize

Prototypes	ReturnCode finalize(	
	const string &enrollDir,	Input
	const string &edbName,	Input
	const string &edbManifest);	Input
Description	This function takes the name of the top-level directory where enrollment database (EDB) and its manifest have	

	been stored. These are described in section 2.3.6. The enrollment directory permissions will be read + write. The function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete. This function should be tolerant of being called two or more times. Second and third invocations should probably do nothing.	
Input Parameters	enrollDir	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
	edbName	The name of a single file containing concatenated templates, i.e. the EDB of section 2.3.6. While the file will have read-write-delete permission, the SDK should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name.
	edbManifest	The name of a single file containing the EDB manifest of section 2.3.6. The file may be opened directly. It is not necessary to prepend a directory name.
Output Parameters	None	
ReturnCode	Success	Success
	FinInputData	Cannot locate the input data - the input files or names seem incorrect.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	FinTemplFormat	One or more template files are in an incorrect format.
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.3.7. The ImageFeatureExtraction Interface

The abstract class ImageFeatureExtraction must be implemented by the SDK developer in a class named exactly SdkImageFeatureExtraction.

	C++ code fragment	Remarks
1.	`class ImageFeatureExtraction`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir) = 0;`
		Initialize the feature extraction session.

4.	`virtual ReturnCode generateIdTemplate(`
	`const MULTIFACE &inputFaces,`
	`PERSONREP &outputTemplate) = 0;`
		This function takes a **MULTIFACE** (see 2.3.3) as input and outputs a proprietary template represented by a **PERSONREP** (see 2.3.10).
		
		For each input image in the **MULTIFACE**, the function shall return the estimated eye centers by setting **PERSONREP**.eyeCoordinates.

5.	`// Destructor`	
6.	`};`	

3.3.7.1. Image feature extraction initialization

Before one or more **MULTIFACE**s are sent to the identification feature extraction function, the test harness will call the initialization function below.

Table 34 – ImageFeatureExtraction::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called once by	

	the NIST application immediately before M ≥ 1 calls to generateIdTemplate. The implementation has read-only access to enrollDir (containing prior enrollment data) and to configDir.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The read-only top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

3.3.7.2. Image feature extraction

A **MULTIFACE** is converted to one identification template using the function below. The result may be stored by NIST, or used immediately. The SDK shall not attempt to store any data.

Table 35 – ImageFeatureExtraction::generateIdTemplate

Prototypes	ReturnCode generateIdTemplate(	
	const **MULTIFACE** &inputFaces,	Input
	PERSONREP &outputTemplate);	Output
Description	This function takes a **MULTIFACE** (see 2.3.3) as input and populates a **PERSONREP** (see 2.3.10) with a proprietary template and eye coordinates. If the function executes correctly, it returns a zero exit status. The NIST calling application may commit the template to permanent storage, or may keep it only in memory (the developer implementation does not need to know). If the function returns a non-zero exit status, the output template will be not be used in subsequent search operations. The function shall not have access to the enrollment data, nor shall it attempt access.	
Input Parameters	inputFaces	An instance of a **Table 12** structure.
Output Parameters	outputTemplate	An instance of a section 2.3.10 class, which stores proprietary template data and eye coordinates. The function shall identify the person's estimated eye centers for each image in the **MULTIFACE**. The eye coordinates shall be captured in the **PERSONREP**.eyeCoordinates variable, which is a vector of **EYEPAIR** objects. In the event the eye centers cannot be calculated, the SDK shall store an **EYEPAIR** and set **EYEPAIR**.isSet to false to indicate there was a failure in generating eye coordinates. In other words, for N images in the **MULTIFACE**.
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of **MULTIFACE**
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	ImgSizeNotSupported	Input image/frame size too small or large
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.3.8. The ImageSearch Interface

The abstract class ImageSearch must be implemented by the SDK developer in a class named exactly SdkImageSearch.

	C++ code fragment	Remarks
1.	`class VideoFeatureExtraction`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir) = 0;`
		Initialize the search session.

4.	`virtual ReturnCode identifyVideo(`
	`const PERSONREP &idTemplate,`
	`const uint32_t candListLength,`
	`CANDIDATELIST &candList) = 0;`
		For video-to-still identification
		
		This function searches a template generated from a **ONEVIDEO** against the enrollment set, and outputs a vector containing candListLength objects of Candidates (see section 2.3.12). Each candidate shall be populated by the implementation and added to candList. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.

5.	`// Destructor`	
6.	`};`	

3.3.8.1. Image identification initialization

The function below will be called once prior to one or more calls of the searching function of Table 37. The function might set static internal variables so that the enrollment database is available to the subsequent identification searches.

Table 36 – ImageSearch::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir);	Input
Description	This function reads whatever content is present in the enrollment_directory, for example a manifest placed there by the ImageFinalize::finalize function.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The read-only top-level directory in which enrollment data was placed.
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

3.3.8.2. Image identification search

The function below performs a video-to-still identification and compares a proprietary identification template generated from a video against the enrollment data and returns a candidate list.

Table 37 – ImageSearch::identifyVideo

Prototype	ReturnCode identifyVideo(	Searches a template generated from a **ONEVIDEO** against the enrollment set (video-to-still)
	const **PERSONREP** &idVideoTemplate,	Input
	const uint32_t candListLength,	Input
	CANDIDATELIST &candList);	Output
Description	This function searches an identification template against the enrollment set, and outputs a vector containing candListLength objects of Candidates (see section 2.3.12). Each candidate shall be populated by the implementation and added to candList. Note that candList will be an empty vector when passed into this function. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.	
Input Parameters	idTemplate	A template from VideoFeatureExtraction::generateIdTemplate() - If the value returned by that function was non-zero the contents of idTemplate will not be used and this function (i.e. identifyVideo) will not be called.
	candListLength	The number of candidates the search should return
Output Parameters	candList	A vector containing candListLength objects of Candidates. The datatype is defined in section 2.3.12. Each candidate shall be populated by the implementation and

		added to this vector. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.
ReturnCode	Success	Success
	IdBadTempl	The input template was defective.
	Vendor	Vendor-defined failure

NOTE: Ordinarily the calling application will set the input candidate list length to operationally typical values, say $0 \leq L \leq 200$, and $L << N$. However, there is interest in the presence of mates much further down the candidate list. We may therefore extend the candidate list length such that L approaches N.

4. References

AN27	NIST Special Publication 500-271: American National Standard for Information Systems — Data Format for the Interchange of Fingerprint, Facial, & Other Biometric Information – Part 1. (ANSI/NIST ITL 1-2007). Approved April 20, 2007.
FRVT 2002	Face Recognition Vendor Test 2002: Evaluation Report, NIST Interagency Report 6965, P. Jonathon Phillips, Patrick Grother, Ross J. Micheals, Duane M. Blackburn, Elham Tabassi, Mike Bone
FRVT 2002b	Face Recognition Vendor Test 2002: Supplemental Report, NIST Interagency Report 7083, Patrick Grother
FRVT 2006	P. Jonathon Phillips, W. Todd Scruggs, Alice J. O'Toole, Patrick J. Flynn, Kevin W. Bowyer, Cathy L. Schott, and Matthew Sharpe. "FRVT 2006 and ICE 2006 Large-Scale Results." NISTIR 7408, March 2007.
FRVT 2013	P. Grother and M. Ngan, Face Recognition Vendor Test (FRVT), Performance of Face Identification Algorithms, NIST Interagency Report 8009, Released May 26, 2014. http://face.nist.gov/frvt
IREX III	P. Grother, G.W. Quinn, J. Matey, M. Ngan, W. Salamon, G. Fiumara, C. Watson, Iris Exchange III, Performance of Iris Identification Algorithms, NIST Interagency Report 7836, Released April 9, 2012. http://iris.nist.gov/irex
ISO STD05	ISO/IEC 19794-5:2005 — Information technology — Biometric data interchange formats — Part 5: Face image data. The standard was published in 2005, and can be purchased from ANSI at http://webstore.ansi.org/ Multipart standard of "Biometric data interchange formats". This standard was published in 2005. It was amended twice to include guidance to photographers, and then to include 3D information. Two corrigenda were published. All these changes and new material is currently being incorporated in revision of the standard. Publication is likely in early 2011. The documentary history is as follows. ISO/IEC 19794-5: Information technology — Biometric data interchange formats — Part 5:Face image data. First edition: 2005-06-15. International Standard ISO/IEC 19794-5:2005 Technical Corrigendum 1: Published 2008-07-01 International Standard ISO/IEC 19794-5:2005 Technical Corrigendum 2: Published 2008-07-01 Information technology — Biometric data interchange formats — Part 5: Face image data AMENDMENT 1: Conditions for taking photographs for face image data. Published 2007-12-15 Information technology — Biometric data interchange formats — Part 5: Face image data AMENDMENT 2: Three dimensional image data. JTC 1/SC37/N3303. FCD text of the second edition. Contact pgrother AT nist DOT gov for more information.
MBE	P. Grother, G .W. Quinn, and P. J. Phillips, Multiple-Biometric Evaluation (MBE) 2010, Report on the Evaluation of 2D Still Image Face Recognition Algorithms, NIST Interagency Report 7709, Released June 22, 2010. Revised August 23, 2010. http://face.nist.gov/mbe
MINEX	P. Grother et al., Performance and Interoperability of the INCITS 378 Template, NIST IR 7296 http://fingerprint.nist.gov/minex04/minex_report.pdf
MOC	P. Grother and W. Salamon, MINEX II - An Assessment of ISO/IEC 7816 Card-Based Match-on-Card Capabilities http://fingerprint.nist.gov/minex/minexII/NIST_MOC_ISO_CC_interop_test_plan_1102.pdf
PERFSTD INTEROP	ISO/IEC 19795-4 — Biometric Performance Testing and Reporting — Part 4: Interoperability Performance Testing. Posted as document 37N2370. The standard was published in 2007. It can be purchased from ANSI at http://webstore.ansi.org/.

Annex A
Submission of Implementations to the FIVE

A.1 Submission of implementations to NIST

NIST requires that all software, data and configuration files submitted by the participants be signed and encrypted. Signing is done with the participant's private key, and encryption is done with the NIST public key. The detailed commands for signing and encrypting are given here: http://www.nist.gov/itl/iad/ig/encrypt.cfm

NIST will validate all submitted materials using the participant's public key, and the authenticity of that key will be verified using the key fingerprint. This fingerprint must be submitted to NIST by writing it on the signed participation agreement.

By encrypting the submissions, we ensure privacy; by signing the submission, we ensure authenticity (the software actually belongs to the submitter). NIST will reject any submission that is not signed and encrypted. NIST accepts no responsibility for anything that is transmitted to NIST that is not signed and encrypted with the NIST public key.

A.2 How to participate

Those wishing to participate in FIVE testing must do all of the following, on the schedule listed on Page 2.

- IMPORTANT: Follow the instructions for cryptographic protection of your SDK and data here. http://www.nist.gov/itl/iad/ig/encrypt.cfm
- Send a signed and fully completed copy of the *Application to Participate in the Face In Video Evaluation (FIVE)*. This is available at http://www.nist.gov/itl/iad/ig/five.cfm. This must identify, and include signatures from, the Responsible Parties as defined in the application. The properly signed FIVE Application to Participate shall be sent to NIST as a PDF.
- Provide an SDK (Software Development Kit) library which complies with the API (Application Programmer Interface) specified in this document.
    - Encrypted data and SDKs below 20MB can be emailed to NIST at five@nist.gov
    - Encrypted data and SDKS above 20MB shall be

      EITHER

        - Split into sections AFTER the encryption step. Use the unix "split" commands to make 9MB chunks, and then rename to include the filename extension need for passage through the NIST firewall.
        - `you%   split -a 3 -d -b 9000000  libFIVE_enron_A_02.tgz.gpg`
        - `you%   ls -1 x??? | xargs -iQ  mv Q libFIVE_enron_A_02_Q.tgz.gpg`
        - Email each part in a separate email. Upon receipt NIST will
        - `nist%  cat FIVE2012_enron_A02_*.tgz.gpg > libFIVE_enron_A_02.tgz.gpg`

      OR

        - Made available as a file.zip.gpg or file.zip.asc download from a generic http webserver[8],

      OR

        - Mailed as a file.zip.gpg or file.zip.asc on CD / DVD to NIST at this address:

FIVE Test Liaison (A203) 100 Bureau Drive A203/Tech225/Stop 8940 NIST Gaithersburg, MD 20899-8940 USA	In cases where a courier needs a phone number, please use NIST shipping and handling on: 301 -- 975 -- 6296.

[8] NIST will not register, or establish any kind of membership, on the provided website.

A.3 Implementation validation

Registered Participants will be provided with a small validation dataset and test program available on the website http://www.nist.gov/itl/iad/ig/five.cfm shortly after the final evaluation plan is released.

The validation test programs shall be compiled by the provider. The output of these programs shall be submitted to NIST.

Prior to submission of the SDK and validation data, the Participant must verify that their software executes on the validation images, and produces correct similarity scores and templates.

Software submitted shall implement the FIVE API Specification as detailed in the body of this document.

Upon receipt of the SDK and validation output, NIST will attempt to reproduce the same output by executing the SDK on the validation imagery, using a NIST computer. In the event of disagreement in the output, or other difficulties, the Participant will be notified.