

CHEX-IA

Child Exploitation Image Analytics FACE RECOGNITION EVALUATION

An Evaluation Activity sponsored by the DHS Science & Technology Directorate



**Homeland
Security**

Science and Technology

Concept, Evaluation Plan and API

Version 0.3, October 22, 2015

Patrick Grother and Mei Ngan

Contact via chexia.face@nist.gov

NIST

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

20

Provisional Timeline of the CHEXIA-FACE Evaluation

Phase 0 API Development	2015-10-26	Draft evaluation plan
	2015-11-06	Final evaluation plan
Phase 1	2015-12-15	Participation starts: Algorithms may be sent to NIST
	2016-01-20	Last day for submission of algorithms to Phase 1
	2016-02-22	Interim results released to Phase 1 participants
Phase 2	2016-04-20	Last day for submission of algorithms to Phase 2
	2016-05-20	Interim results released to Phase 2 participants
Phase 3	2016-07-27	Last day for submission of algorithms to Phase 3
	2016-Q4	Release of final public report

21

22

23

24

Notable differences from FRVT 2012

25 Please note that this document is derived from the FRVT 2012 API document for continuity and to aid implementers of
26 the CHEXIA-FACE API.

- 27 — This evaluation is dedicated solely to imagery relevant to the child exploitation. NIST seeks to assist developers in any
28 way possible to improve algorithms on this task, and is open to creative ideas on how to do so.
- 29 — We anticipate running the algorithms only on child exploitation imagery. We may also run algorithms on other
30 images if that will isolate relevant factors that will influence accuracy. We do not intend to run the algorithms on
31 cooperative images used in recent FRVT tests.
- 32 — This evaluation drops the following:
 - 33 — Facial age, gender, pose conformance, and expression estimation for still images (see section 1.9)
 - 34 — The class F evaluation of frontal pose rendering algorithms
- 35 — This evaluation:
 - 36 — Merges the idea of “still” and “video”. This abstraction supports verification and identification functions
37 where either “sample” may be a still or video. See section 2.3.2.
 - 38 — Adds a face detection tasks in which the algorithm reports locations of faces detected in images. See Section
39 3.3.
 - 40 — Adds a clustering task in which the algorithm finds and groups images of an unknown number of identities.
41 See section 0.
 - 42 — Employs GPUs on some NIST machines.
- 44 — The header/source files for the API will be made available to implementers at <http://nigos.nist.gov:8080/chexia-face>.

45

46 **Table of Contents**

47 1. CHEXIA-FACE 5

48 1.1. Scope 5

49 1.2. Audience 5

50 1.3. Market drivers 5

51 1.4. Test datasets 6

52 1.5. Offline testing 6

53 1.6. Phased testing 6

54 1.7. Interim reports 7

55 1.8. Final reports 7

56 1.9. Application scenarios 7

57 1.10. Options for participation 8

58 1.11. Number and schedule of submissions 8

59 1.12. Core accuracy metrics 9

60 1.13. Reporting template size 9

61 1.14. Reporting computational efficiency 9

62 1.15. Exploring the accuracy-speed trade-space 9

63 1.16. Hardware specification 9

64 1.17. Operating system, compilation, and linking environment 10

65 1.18. Software and Documentation 10

66 1.19. Runtime behavior 12

67 1.20. Threaded computations 12

68 1.21. Time limits 13

69 1.22. Ground truth integrity 13

70 2. Data structures supporting the API 13

71 2.1. Overview 13

72 2.2. Requirement 13

73 2.3. File formats and data structures 13

74 2.4. File structures for enrolled template collection 18

75 3. API Specification 19

76 3.1. 1:1 Verification 19

77 3.2. 1:N Identification 21

78 3.3. Face Detection 26

79 3.4. Clustering 27

80 4. References 28

81 Annex A Submission of Implementations to the CHEXIA-FACE 29

82 A.1 Submission of implementations to NIST 29

83 A.2 How to participate 29

84 A.3 Implementation validation 30

86 **List of Tables**

87 Table 1 – Main image corpora (others will be used) 6

88 Table 2 – Subtests supported under the CHEXIA-FACE activity 7

89 Table 3 – CHEXIA-FACE classes of participation 8

90 Table 4 – Cumulative total number of algorithms, by class 8

91 Table 5 – Implementation library filename convention 11

92 Table 6 – Number of threads allowed for each application 12

93 Table 7 – Processing time limits in milliseconds, per 640 x 480 image 13

94 Table 8 – Structure for a single image or video frame 14

95 Table 9 – Structure for a set of images or video frames from a single person 15

96 Table 10 – Labels describing categories of Multifaces 15

97 Table 11 – Structure for a pair of eye coordinates 15

CHEXIA-FACE

98 Table 12 – PersonTrajectory typedef 16

99 Table 13 – Structure for bounding box around a detected face 16

100 Table 14 – Structure for a single hypothesized cluster membership 17

101 Table 15 – Structure for hypothesized cluster membership for face(s) in an image 17

102 Table 16 – Structure for a candidate 17

103 Table 17 – Return codes for API function calls 18

104 Table 18 – Enrollment dataset template manifest 18

105 Table 19 – Functional summary of the 1:1 application 19

106 Table 20 – Initialization 19

107 Table 21 – Template generation 19

108 Table 22 – Template matching 20

109 Table 23 – Procedural overview of the identification test 21

110 Table 24 – Enrollment initialization 22

111 Table 25 – Enrollment feature extraction 23

112 Table 26 – Enrollment finalization 23

113 Table 27 – Identification feature extraction initialization 24

114 Table 28 – Identification feature extraction 24

115 Table 29 – Identification initialization 25

116 Table 30 – Identification search 25

117 Table 31 – SDK initialization 26

118 Table 32 – Face detection 26

119 Table 33 – SDK initialization 27

120 Table 34 – Clustering 27

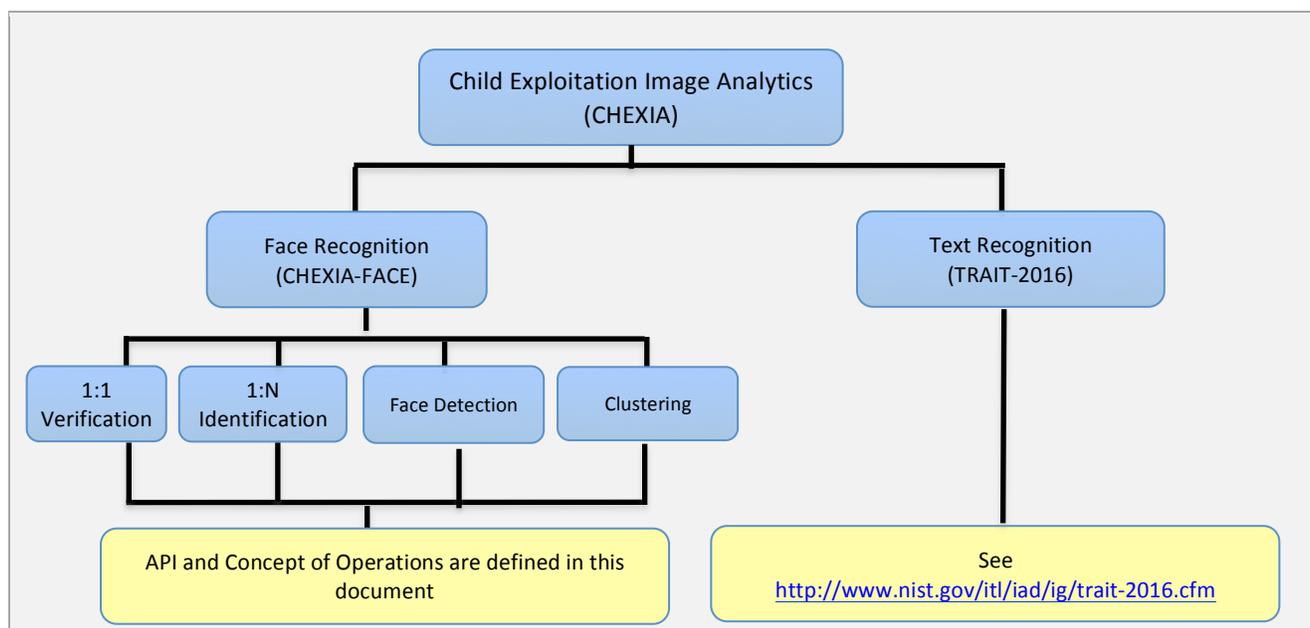
121

122

123 1. CHEXIA-FACE

124 1.1. Scope

125 This document establishes a concept of operations and an application programming interface (API) for evaluation of face
 126 recognition (FR) implementations submitted to NIST's Child Exploitation Image Analytics Face Recognition Evaluation
 127 (CHEXIA-FACE).



128

129 1.2. Audience

130 Universities and commercial entities with capabilities in any of the following areas are invited to participate in the
 131 CHEXIA-FACE test.

- 132 – Identity verification with face recognition algorithms.
- 133 – Large scale identification implementations.
- 134 – Face detection algorithms.
- 135 – Implementations with an ability to cluster (find and group) images of an unknown number of identities.

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

139 1.3. Market drivers

140 There is a growing market around digital forensics – the ability to extract semantic information from imagery that is useful
 141 to an investigation. This test specifically is intended to assess the efficacy of face recognition algorithms on child
 142 exploitation imagery. These images are of interest to NIST's partner law enforcement agencies that seek to employ face
 143 recognition in investigating this area of serious crime. The primary applications are identification of previously known
 144 victims and suspects, detection of new victims and suspects. Given a collection of images, produce a cluster of identities,
 145 from which (law enforcement) investigations can proceed.

146 A parallel effort, TRAIT 2016, seeks to improve the capability of algorithms to recognize text in unconstrained images.
 147 Text appears frequently in child exploitation imagery. See <http://www.nist.gov/itl/iad/ig/trait-2016.cfm>.

148

149 1.4. Test datasets

150 NIST anticipates running the algorithms only on child exploitation imagery. NIST may also run algorithms on other images
 151 if that will isolate factors that will influence accuracy. NIST does not intend to run the algorithms on cooperative images
 152 used in recent FRVT tests. The data has, in some cases, been estimated from initial small partitions. The completion of
 153 this section depends on further work. The information is subject to change.

154 **Table 1 – Main image corpora (others may be used)**

	Child exploitation	TBD
Collection, environment	Mostly inside a home, sometimes outdoors	
Live photo, Paper scan	Live	
Documentation	See NOTE 1	
Compression from [MBE 2010] ¹	Variable	
Maximum image size	Some from contemporary SLR camera, some 3000x4000 and higher.	
Minimum image size	240 x 240	
Eye to eye distance pixels	20 to 1000 approximately	
Pose	The images have compound pitch and yaw rotations.	
Full frontal geometry	Rarely	
Intended use	All CHEXIA-FACE tasks	
Age	Many below 10, some to age 18. Rarely an adult.	

155

156 **Child exploitation images:** These images are of interest to NIST's partner law enforcement agencies who seek to employ
 157 face recognition in investigating this area of serious crime. The primary applications are identification of previously
 158 known victims and suspects, detection of new victims and suspects, and clustering of all individuals across entire image
 159 collections.

160 These images are illicit pornographic images and video. The images are present on digital media seized in criminal
 161 investigations. The files include children who range in age from infant through adolescent. In addition a few adult faces
 162 sometimes occur also. Some of the images are innocuous "family photographs". The majority, however, feature coercion,
 163 abuse, and sexual activity.

164 From a face recognition viewpoint, the images will be difficult for the following reasons (in order): highly variable pose
 165 (including adverse compound pitch and yaw); occlusion (by hair, other persons, body parts and objects); variable and
 166 directional lighting; evidence that face recognition in children is difficult even with cooperative photographs – see
 167 Annex B below which excerpts [NIST8009].

168 1.5. Offline testing

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

174 1.6. Phased testing

175 To support development, CHEXIA-FACE will run in three phases. In each phase, NIST will evaluate implementations on a
 176 first-come-first-served basis and will return results to providers as expeditiously as possible. The final phase, which will
 177 result in public reports. Providers may not submit revised algorithms to NIST until NIST provides results for the prior
 178 phase. The frequency with which a provider may submit implementations to NIST will depend on the times needed for
 179 developer preparation, transmission to NIST, validation, execution and scoring at NIST, and developer review and decision
 180 processes.

¹ Compression effects were studied under MBE 2010 in NIST Interagency Report 7830, linked from <http://face.nist.gov/mbc>

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

182 **1.7. Interim reports**

183 The performance of each implementation will be reported in a "score-card". This will be provided to the participant. It is
 184 intended to facilitate development. Score cards will: be machine generated (i.e. scripted); be provided to participants
 185 with identification of their implementation, include timing, accuracy and other performance results, include results from
 186 other implementations, but will not identify the other providers; be expanded and modified as revised implementations
 187 are tested, and as analyses are implemented; be produced independently of the other status of other providers'
 188 implementations; be regenerated on-the-fly, usually whenever any implementation completes testing, or when new
 189 analysis is added.

190 NIST does not intend to release these test reports publicly. NIST may release such information to the U.S. Government
 191 test sponsors; NIST will request that agencies not release this content.

192 **1.8. Final reports**

193 NIST will publish one or more final public reports. NIST may also publish: additional supplementary reports (typically as
 194 numbered NIST Interagency Reports); in other academic journals; in conferences and workshops (typically PowerPoint).

195 Our intention is that the final test reports will publish results for the best-performing implementation from each
 196 participant. Because "best" is under-defined (accuracy vs. time vs. template size, for example), the published reports may
 197 include results for other implementations. The intention is to report results for the most capable implementations (see
 198 section 1.12, on metrics). Other results may be included (e.g. in appendices) to show, for example, examples of progress
 199 or tradeoffs. **IMPORTANT:** Results will be attributed to the providers.

200 **1.9. Application scenarios**

201 The test will include one-to-one verification tests and one-to-many identification tests³ [MBE 2010, IREX III, NIST8009] for
 202 still images and video clips. As described in Table 2, the test is intended to represent:

- 203 – Close-to-operational use of face recognition technologies in identification applications in which the enrolled dataset
 204 could contain images in the hundreds of thousands.
- 205 – Verification scenarios in which samples are compared.
- 206 – Face detection in stills and videos with one or more persons in the sample.
- 207 – Grouping (clustering) identities in mixed media

208 **Table 2 – Subtests supported under the CHEXIA-FACE activity**

#	A	C	D	G	
1.	Aspect	1:1 verification	1:N identification	Detection	Clustering
2.	Enrollment dataset	None (applies to single samples; there is no concept of gallery or enrollment database)	N enrolled subjects	None, application to single images	The concepts of enrollment and search sets do not exist
3.	Prior NIST test references	Equivalent to 1 to 1 matching in [MBE 2010]	Equivalent to 1 to N matching in [NIST 8009]		
4.	Example application	Verification of e-Passport facial image against a live border-crossing image.	Open-set identification of an image against a central database, e.g. a search of a mugshot against a database of known criminals.	Often used in conjunction with face recognition; also used in video surveillance, human computer interaction, and image database management	Group of individuals whose faces are present in a collection e.g. a studio archive; e.g. media seized in an arrest.
5.	Score or feature space normalization support	If any, normalization techniques are only possible over datasets internal to the	Any score or feature based statistical normalization techniques-are applied		Any score or feature based statistical normalization techniques-are applied

CHEXIA-FACE

		implementation.	against enrollment database		internally
6.	Intended number of subjects	Up to $O(10^4)$	Up to $O(10^5)$ but dependence on N will be computed. From $O(10^2)$ upwards.	Expected $O(10^4)$	Expected $O(10^3)$
7.	Number of images per individual	Variable: one or more still images, or a video clip	Variable: one or more still images, or a video clip	Variable	Variable
8.	Number of persons in one sample	1	1 or more persons	1 or more persons	1 or more persons

209

210 NOTE 1: The vast majority of images are color. The API supports both color and greyscale images.

211 NOTE 2: For the operational datasets, it is not known what processing was applied to the images before they were
 212 archived. So, for example, we do not know whether gamma correction was applied. NIST considers that best practice,
 213 standards and operational activity in the area of image preparation remains weak.

214 **1.10. Options for participation**

215 The following rules apply:

- 216 – A participant must properly follow, complete and submit the Annex A Participation Agreement. This must be done
 217 once, not before January 1, 2016. It is not necessary to do this for each submitted implementation.
- 218 – All participants shall submit at least one class D algorithm.
- 219 – Class A (1:1) algorithms may be submitted only if at least 1 class D (detection) algorithm is also submitted.
- 220 – Class C (1:N) algorithms may be submitted only if at least 1 class A (1:1) algorithm is also submitted.
- 221 – Class G (clustering) algorithms may be submitted only if at least 1 class C (1:N) algorithm is also submitted.
- 222 – Class D (detection) algorithms may be submitted alone, without submission to any other classes of participation.
- 223 – All submissions shall implement exactly one of the functionalities defined in Table 3. A library shall not implement
 224 the API of more than one class.

225

Table 3 – CHEXIA-FACE classes of participation

Function	1:1 verification	1:N identification	Detection	Clustering
Class label	A	C	D	G
Co-requisite class	D	D + A	None	D + A + C
API requirements	3.1	0	3.3	0

226 Class A might be preferred by academic institutions because the API is simple, supporting just the elemental hypothesis
 227 test: "are the images from the same person or not?"

228 **1.11. Number and schedule of submissions**

229 The test is conducted in three phases, as scheduled on page 2. The maximum total (i.e. cumulative) number of
 230 submissions is regulated in Table 4.

231

Table 4 – Cumulative total number of algorithms, by class

#	Phase 1	Total over Phases 1 + 2	Total over Phases 1 + 2 + 3
Class A : Verification	2	4	6 if at least 1 was successfully executed by end Phase 1 2 otherwise
Class C : Identification	2	4	6 if at least 1 was successfully executed by end Phase 1 2 otherwise
Class D : Detection	2	2	3 if at least 1 was successfully executed by end Phase 1 1 otherwise

Class G : Clustering	2	2	4 if at least 1 was successfully executed by end Phase 1 2 otherwise
----------------------	---	---	---

232 The numbers above may be increased as resources allow.

233 NIST cannot conduct surveys over runtime parameters – essentially to limit the extent to which participants are able to
234 train on the test data.

235 **1.12. Core accuracy metrics**

236 Notionally the error rates for verification applications will be false match and false non-match error rates, FMR and FNMR.
237 These will be modified to include the effects of failure to make a template.

238 For identification testing, the test will target open-universe applications such as watch-lists. It will not address the closed-
239 set task because it is operationally uncommon. Metrics include false positive and negative identification rate (FPIR and
240 FNIR) that depend on threshold and rank.

241 Rank-based metrics are appropriate for one-to-many applications that employ human examiners to adjudicate candidate
242 lists. Score based metrics are appropriate for cases where transaction volumes are too high for human adjudication or
243 when false alarm rates must otherwise be low. See [NIST8009].

244 **1.13. Reporting template size**

245 Because template size is influential on storage requirements and computational efficiency, this API supports
246 measurement of template size. NIST will report statistics on the actual sizes of templates produced by face recognition
247 implementations submitted to CHEXIA-FACE. NIST may report statistics on runtime memory usage. Template sizes were
248 reported in the FRVT 2012 test², IREX III test³ and the MBE-STILL 2010 test⁴.

249 **1.14. Reporting computational efficiency**

250 As with other tests, NIST will compute and report recognition accuracy. In addition, NIST will also report timing statistics
251 for all core functions of the submitted implementations. This includes feature extraction, 1:1 and 1:N recognition,
252 detection, and clustering. For an example of how efficiency can be reported, see the final report of the FRVT 2012 test
253 [NIST8009], IREX III test³ and the MBE-STILL 2010 test⁴.

254 Note that face recognition applications optimized for pipelined 1:N searches may not demonstrate their efficiency in pure
255 1:1 comparison applications.

256 **1.15. Exploring the accuracy-speed trade-space**

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

262 **1.16. Hardware specification**

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

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

² See the FRVT 2012 test report: NIST Interagency Report 8009, linked from <http://face.nist.gov/frvt>

³ See the IREX III test report: NIST Interagency Report 7836, linked from <http://iris.nist.gov/irex>

⁴ See the MBE-STILL 2010 test report, NIST Interagency Report 7709, linked from <http://face.nist.gov/mbe>

- 267 • Dell M905 - Quad AMD Opteron 8376HE 2 GHz CPUs⁵ (4 cores each)
- 268 • Dell M910 - Dual Intel Xeon X7560 2.3 GHz CPUs (8 cores each)
- 269 • Dual Intel Xeon E5-2695 3.3 GHz CPUs (14 cores each) with Dual NVIDIA Tesla K40 GPUs

270 NOTE: Implementations must be functional on machines with and without GPU capability.

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

273 NIST is requiring use of 64 bit implementations throughout. This will support large memory allocation to support 1:N
274 identification task. For video, given the data expectations and the occurrence of faces in the imagery, we anticipate the
275 developers will have sufficient memory for video templates. Note that while the API allows read access of the disk during
276 the 1:N search, the disk is, of course, relatively slow.

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

280 1.17. Operating system, compilation, and linking environment

281 The operating system that the submitted implementations shall run on will be released as a downloadable file accessible
282 from <http://nigos.nist.gov:8080/evaluations/> which is the 64-bit version of CentOS 7 running Linux kernel 3.10.0.

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

285 NIST will link the provided library file(s) to our C++ language test drivers. Participants are required to provide their library
286 in a format that is linkable using the C++11 compiler, g++ version 4.8.2.

287 A typical link line might be

288 `g++ -l. -Wall -m64 -o chexiaface chexiaface.cpp -L. -lchexiaface_Enron_A_07`

289 The Standard C++ library should be used for development. The prototypes from this document will be written to a file
290 "chexiaface.h" which will be included via

```
#include <chexiaface.h>
```

291 The header files will be made available to implementers at <http://nigos.nist.gov:8080/chexia-face/>

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

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

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

299 1.18. Software and Documentation

300 1.18.1. Library and Platform Requirements

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

⁵ `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`

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

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

310 **Table 5 – Implementation library filename convention**

Form	libCHEXIAFACE_provider_class_sequence.ending				
Underscore delimited parts of the filename	libCHEXIAFACE	provider	class	sequence	ending
Description	First part of the name, required to be this.	Single word name of the main provider EXAMPLE: Acme	Function classes supported in Table 3. EXAMPLE: C	A two digit decimal identifier to start at 00 and increment by 1 every time a library is sent to NIST. EXAMPLE: 07	.so
Example	libCHEXIAFACE_Acme_C_07.so				

311
 312 NIST will report the size of the supplied libraries.

313 **1.18.2. Configuration and developer-defined data**

314 The implementation under test may be supplied with configuration files and supporting data files. NIST will report the
 315 size of the supplied configuration files.

316 **1.18.3. Submission folder hierarchy**

- 317 Participant submissions should contain the following folders at the top level
- 318 • lib/ - contains all participant-supplied software libraries
 - 319 • config/ - contains all configuration and developer-defined data
 - 320 • doc/ - contains any participant-provided documentation regarding the submission

321 **1.18.4. Installation and Usage**

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

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

327 The implementation shall not use nor enforce any usage controls or limits based on licenses, number of executions,
 328 presence of temporary files, etc. It shall remain operable with no expiration date.

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

330 **1.18.5. Documentation**

331 Participants shall provide documentation of additional functionality or behavior beyond that specified here. The
 332 documentation must define all (non-zero) developer-defined error or warning return codes.

333 **1.18.6. Modes of operation**

334 Implementations shall not require NIST to switch “modes” of operation or algorithm parameters. For example, the use of
 335 two different feature extractors must either operate automatically or be split across two separate library submissions.

336 **1.19. Runtime behavior**

337 **1.19.1. Interactive behavior, stdout, logging**

338 The implementation will be tested in non-interactive “batch” mode (i.e. without terminal support). Thus, the submitted
339 library shall:

- 340 – Not use any interactive functions such as graphical user interface (GUI) calls, or any other calls which require
341 terminal interaction e.g. reads from “standard input”.
- 342 – Run quietly, i.e. it should not write messages to "standard error" and shall not write to “standard output”.
- 343 – If requested by NIST for debugging, include a logging facility in which debugging messages are written to a log file
344 whose name includes the provider and library identifiers and the process PID.

345 **1.19.2. Exception Handling**

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

348 **1.19.3. External communication**

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

354 **1.19.4. Stateless behavior**

355 All components in this test shall be stateless, except as noted. This applies to face detection, feature extraction and
356 matching. Thus, all functions should give identical output, for a given input, independent of the runtime history. NIST
357 will institute appropriate tests to detect stateful behavior. If detected, NIST will take appropriate steps, including but not
358 limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and
359 documentation of the activity in published reports.

360 **1.20. Threaded computations**

361 Table 6 shows the limits on the numbers of threads an implementation may use for each of the classes of participation. In
362 many cases threading is not permitted (i.e. T=1) because NIST will parallelize the test by dividing the workload across
363 many cores and many machines.

364 **Table 6 – Number of threads allowed for each application**

	A	C	D	G
Function	1:1 verification	1:N identification	Detection	Clustering
Feature extraction	1	1	1	1 ≤ T ≤ 16
Verification	1	NA		
Finalize enrollment (before 1:N)	NA	1 ≤ T ≤ 16		
Identification	NA	1		

365 For comparative timing, the IREX III³ test report estimated a factor by which the speed of threaded algorithms would be
366 adjusted. Non-threaded implementations will eliminate the need for NIST to apply such techniques [IREX III].

367 NIST will not run an implementation from participant X and an implementation from participant Y on the same machine at
368 the same time.

369 To expedite testing, for single-threaded libraries, NIST will run up to P = 16 processes concurrently. NIST's calling
370 applications are single-threaded.

371 **1.21. Time limits**

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

376 The time limits apply per image. When K images of a person are present, the time limits shall be increased by a factor K.

377 **Table 7 – Processing time limits in milliseconds, per 640 x 480 image**

	A	C	D	G
Function	1:1 verification	1:N identification	Detection	Clustering
Feature extraction enrollment	1000 (1 core) 600x480 pixels	1000 (1 core) 600x480 pixels	K*500 (1 core), where K=number of persons in the image	1000 (1 core)
Feature extraction for verification or identification	1000 (1 core) 600x480 pixels	1000 (1 core) 600x480 pixels		
Verification	5 (1 core)	NA		
Identification of one search image against 1,000,000 single-image Multiface records.	NA	10000 (16 cores) or 160000 (1 core)		
Enrollment finalization of 1,000,000 single-image Multiface records (including disk IO time)	NA	7,200,000 (up to 16 cores)	NA	NA

378 **1.22. Ground truth integrity**

379 Some of the test data is derived from operational systems and may contain ground truth errors in which

- 380 – a single person is present under two different identifiers, or
- 381 – two persons are present under one identifier, or
- 382 – in which a face is not present in the image.

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

388 **2. Data structures supporting the API**

389 **2.1. Overview**

390 This section describes separate APIs for the core face recognition applications described in section 1.9. All submissions to
 391 CHEXIA-FACE shall implement the functions required by the rules for participation listed before Table 3.

392 **2.2. Requirement**

393 CHEXIA-FACE participants shall implement the relevant C++ prototyped interfaces of clause 3. C++ was chosen in order to
 394 make use of some object-oriented features.

395 **2.3. File formats and data structures**

396 **2.3.1. Overview**

397 In this face recognition test, an individual is represented by $K \geq 1$ two-dimensional facial images, and by subject and
 398 image-specific metadata.

399 **2.3.2. Data structures for encapsulating multiple images or video frames**

400 Some of the proposed datasets includes $K > 2$ images per person for some persons. This affords the possibility to model a
 401 recognition scenario in which a new image of a person is compared against all prior images⁶. Use of multiple images per
 402 person has been shown to elevate accuracy over a single image [MBE 2010].

403 For still-face recognition in this test, NIST will enroll $K \geq 1$ images under each identity. Normally the probe will consist of a
 404 single image, but NIST may examine the case that it could consist of multiple images. Ordinarily, the probe images will be
 405 captured after the enrolled images of a person⁷. The method by which the face recognition implementation exploits
 406 multiple images is not regulated: The test seeks to evaluate developer provided technology for multi-presentation fusion.

407 This document defines a template to be the result of applying feature extraction to a set of $K \geq 1$ images. An algorithm
 408 might internally fuse K feature sets into a single model or maintain them separately - In any case the resulting proprietary
 409 template is contained in a contiguous block of data. All verification and identification functions operate on such multi-
 410 image templates.

411 The number of images per person will vary, and images may not be acquired uniformly over time. NIST currently
 412 estimates that the number of images will never exceed 100.

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

417 For the CHEXIA-FACE API, K images of an individual are contained in data structure of Table 13. Each file contains a
 418 standardized image format, e.g. PNG (lossless) or JPEG (lossy).

419

420 NOTE: For the 1:1 verification task, all images will contain one and only one face. For all other CHEXIA-FACE tasks, images
 421 in the test may contain one or more faces in an image.

422

423 **Table 8 – Structure for a single image or video frame**

	C++ code fragment	Remarks
1.	struct Image	
2.	{	
3.	uint16_t width;	Number of pixels horizontally
4.	uint16_t height;	Number of pixels vertically
5.	uint16_t depth;	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 I I I I I I I I
8.	};	

424

425

426

427

⁶ For example, if a banned driver applies for a driving license under a new name, and the local driving license authority maintains a driving license system in which all previous driving license photographs are enrolled, then the fraudulent application might be detected if the new image matched any of the prior images. This example implies one (elemental) method of using the image history.

⁷ To mimic operational reality, NIST intends to maintain a causal relationship between probe and enrolled images. This means that the enrolled images of a person will be acquired before all the images that comprise a probe.

428

Table 9 – Structure for a set of images or video frames

	C++ code fragment	Remarks
1.	<code>struct Multiface</code>	
2.	<code>{</code> <code> typedef std::vector<Image> images;</code>	Vector containing F pre-allocated face images. The number of items is accessible via the <code>vector::size()</code> function.
3.	<code> MultifaceLabel description;</code>	Single description of the <code>Multiface</code> . The allowed values for this field are specified in the enumeration in Table 10.
4.	<code> uint16_t framesPerSec;</code>	The frame rate of the video sequence in frames-per-second. Only defined if <code>description==Video</code> ; otherwise set to -1.
5.	<code>};</code>	

429

430

431

A **Multiface** will be accompanied by one of the labels given below. Face recognition implementations submitted to CHEXIA-FACE should tolerate **Multifaces** of any category.

432

Table 10 – Labels describing categories of Multifaces

	Label as C++ enumeration	Meaning
1.	<code>enum class MultifaceLabel {</code>	
2.	<code> Unknown=0,</code>	Either the label is unknown or unassigned.
3.	<code> Fixed=1,</code>	Images are still photos from a non-moving camera.
4.	<code> Event=2,</code>	Images are still photos from the same event, possibly from several cameras or with camera movement.
5.	<code> Group=3,</code>	Still photos from one person on arbitrary occasions.
6.	<code> Video=4</code>	Images are a sequence of video frames.
7.	<code>};</code>	

433

2.3.3. Data structure for eye coordinates

435

436

437

Implementations should return eye coordinates of each facial image. This function, while not necessary for a recognition test, will assist NIST in assuring the correctness of the test database. The primary mode of use will be for NIST to inspect images for which eye coordinates are not returned, or differ between implementations.

438

439

The eye coordinates shall follow the placement semantics of the ISO/IEC 19794-5:2005 standard - the geometric midpoints of the endocanthion and exocanthion (see clause 5.6.4 of the ISO standard).

440

Sense: The label "left" refers to subject's left eye (and similarly for the right eye), such that `xright < xleft`.

441

Table 11 – Structure for a pair of eye coordinates

	C++ code fragment	Remarks
1.	<code>struct EyePair</code>	
2.	<code>{</code>	
	<code> bool isAssigned;</code>	If the eye coordinates have been computed and assigned successfully, this value should be set to true, otherwise false.
3.	<code> int16_t xleft;</code>	X and Y coordinate of the center of the subject's left eye. Out-of-range values (e.g. <code>x < 0</code> or <code>x >= width</code>) indicate the implementation believes the eye center is outside the image.
4.	<code> int16_t yleft;</code>	
5.	<code> int16_t xright;</code>	X and Y coordinate of the center of the subject's right eye. Out-of-range values (e.g. <code>x < 0</code> or <code>x >= width</code>) indicate the implementation believes the eye center is outside the image.
5.1.	<code> int16_t yright;</code>	
6.	<code> uint16_t frameNum</code>	For <code>Multifaces</code> where <code>description==Video</code> , this would be the frame number that corresponds to the video frame from which the eye coordinates were generated. (i.e., the i-th frame from the video sequence). This field should not be set for eye coordinates for a single still image.
7.	<code>};</code>	

442 **2.3.4. Data type for representing eye coordinates from a Multiface**

443 **Table 12 – PersonTrajectory typedef**

	C++ code fragment	Remarks
1.	<code>using PersonTrajectory = std::vector<EyePair>;</code>	<p>Vector of <code>EyePair</code> objects for a <code>Multiface</code> where eyes were detected. This data structure should store eye coordinates for each video frame or image where eyes were detected for a particular person. For <code>Multifaces</code> where the person’s eyes were not detected, the SDK shall not add an <code>EyePair</code> to this data structure.</p> <p>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.</p>

444 **2.3.5. Class for representing a person detected in a Multiface**

	C++ code fragment	Remarks
1.	<code>class PersonRep</code>	
2.	<code>{</code>	
	<code>private:</code>	
3.	<code>PersonTrajectory eyeCoordinates;</code>	Data structure for capturing eye coordinates
4.	<code>PersonTemplate proprietaryTemplate;</code>	<code>PersonTemplate</code> is a wrapper to a <code>uint8_t*</code> for capturing proprietary template data representing a person from a <code>Multiface</code> .
5.	<code>public:</code>	
6.	<code>PersonRep(const uint64_t inSize);</code>	The constructor takes a size parameter and allocates memory of <code>inSize</code> . <code>getPersonTemplatePtr()</code> 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.	<code>void pushBackEyeCoord(const EyePair &eyes);</code>	This function should be used to add <code>EyePairs</code> for the video frames or images where eye coordinates were detected.
8.	<code>uint8_t* getPersonTemplatePtr();</code>	This function returns a <code>uint8_t*</code> to the template data.
9.	<code>uint64_t getPersonTemplateSize() const;</code>	This function returns the size of the template data.
10.	<code>//... getter methods, copy constructor,</code>	
	<code>//... assignment operator</code>	
11.	<code>};</code>	

445 **2.3.6. Data Structure for detected face**

446 For face detection, implementations shall return bounding box coordinates of each detected face in an image. See
 447 section 3.3 for API details.

448 **Table 13 – Structure for bounding box around a detected face**

	C++ code fragment	Remarks
1.	<code>struct BoundingBox</code>	x-coordinate of top-left corner of bounding box around face
	<code>{</code>	
2.	<code>uint16_t x;</code>	y-coordinate of top-left corner of bounding box around face
	<code>uint16_t y;</code>	width, in pixels, of bounding box around face
3.	<code>uint16_t width;</code>	height, in pixels, of bounding box around face
4.	<code>uint16_t height;</code>	
5.	<code>};</code>	

449 **2.3.7. Data Structure for hypothesized cluster membership**

450 For clustering, implementations shall assign image samples to clusters using the structure of Table 14. See section 0 for
 451 API details.

452 **Table 14 – Structure for a single hypothesized cluster membership**

	C++ code fragment	Remarks
1.	<code>struct ClusterMember</code>	
2.	<code>{</code>	
3.	<code>uint32_t clusterId;</code>	Positive integer assigned by the implementation indicating a cluster label. All images of a person should share this same integer.
4.	<code>double similarityScore;</code>	Measure of similarity between the input sample and other images of the same cluster (presumably of the same person). This score can be used to perform threshold-based analysis of algorithm performance.
5.	<code>BoundingBox face;</code>	Bounding box coordinates corresponding to the face/identity belonging to <code>clusterId</code> from the image.
6.	<code>};</code>	

453 **Table 15 – Structure for hypothesized cluster membership for face(s) in an image**

	C++ code fragment	Remarks
1.	<code>struct ClusterMembersInImage</code>	
2.	<code>{</code>	
3.	<code>int32_t failed;</code>	This value should be set as follows: 0 if the input sample was processed successfully 2 if no faces could be detected in the input 4 if the software failed 6 if the software detects malformed input
4.	<code>std::vector<ClusterMember> members;</code>	For each person found in an image, the list of hypothesized cluster membership(s).
5.	<code>};</code>	

454 **2.3.8. Data structure for result of an identification search**

455 All identification searches shall return a candidate list of a NIST-specified length. The list shall be sorted with the most
 456 similar matching entries list first with lowest rank. The data structure shall be that of Table 16. See section 0 for API
 457 details.

458 **Table 16 – Structure for a candidate**

	C++ code fragment	Remarks
1.	<code>struct Candidate</code>	
2.	<code>{</code>	
3.	<code>bool isAssigned;</code>	If the candidate is valid, this should be set to true. If the candidate computation failed, this should be set to false.
4.	<code>std::string templateId;</code>	The Template ID from the enrollment database manifest defined in clause 2.4.
5.	<code>double similarityScore;</code>	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.	<code>};</code>	

459

460

461 **2.3.9. Enumeration of return codes for API function calls**

462 **Table 17 – Return codes for API function calls**

	Return code as C++ enumeration	Meaning
	enum class ReturnCode {	
1.	Success=0,	Success
2.	ConfigError=1,	Error reading configuration files
3.	RefuseInput=2,	Elective refusal to process the input
4.	ExtractError=3,	Involuntary failure to process the image
5.	ParseError=4,	Cannot parse the input data
6.	TemplateCreationError=5,	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
7.	VerifTemplateError=6,	For matching, either or both of the input templates were result of failed feature extraction
8.	EnrollDirError=7,	An operation on the enrollment directory failed (e.g. permission, space)
9.	NumDataError=8,	The SDK cannot support the number of persons or images
10.	TemplateFormatError=9,	One or more template files are in an incorrect format or defective
11.	InputLocationError=10,	Cannot locate the input data - the input files or names seem incorrect
12.	VendorError=11	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.
13.	};	

463 **2.3.10. Data type for similarity scores**

464 Identification and verification functions shall return a measure of the similarity between the face data contained in the
 465 two templates. The datatype shall be an eight byte double precision real. The legal range is [0, DBL_MAX], where the
 466 DBL_MAX constant is larger than practically needed and defined in the <limits.h> include file. Larger values indicate more
 467 likelihood that the two samples are from the same person.

468 Providers are cautioned that algorithms that natively produce few unique values (e.g. integers on [0,127]) will be
 469 disadvantaged by the inability to set a threshold precisely, as might be required to attain a false match rate of exactly
 470 0.0001, for example.

471 **2.4. File structures for enrolled template collection**

472 An SDK converts a `Multiface` into a template, using, for example the
 473 `convertMultifaceToEnrollmentTemplate` function of section 3.2.2.2. To support the class C identification
 474 functions of Table 3, NIST will concatenate enrollment templates into a single large file, the EDB (for enrollment
 475 database). The EDB is a simple binary concatenation of proprietary templates. There is no header. There are no
 476 delimiters. The EDB may be hundreds of gigabytes in length.

477 This file will be accompanied by a manifest; this is an ASCII text file documenting the contents of the EDB. The manifest
 478 has the format shown as an example in Table 18. If the EDB contains N templates, the manifest will contain N lines. The
 479 fields are space (ASCII decimal 32) delimited. There are three fields. Strictly speaking, the third column is redundant.

480 Important: If a call to the template generation function fails, or does not return a template, NIST will include the Template
 481 ID in the manifest with size 0. Implementations must handle this appropriately.

482 **Table 18 – Enrollment dataset template manifest**

Field name	Template ID	Template Length	Position of first byte in EDB
Datatype required	std::string	Unsigned decimal integer	Unsigned decimal integer
Datatype length required		4 bytes	8 bytes
Example lines of a manifest file appear to the right. Lines 1, 2, 3 and N appear.	90201744	1024	0
	person01	1536	1024
	7456433	512	2560
	...		
	subject12	1024	307200000

483
484 The EDB scheme avoids the file system overhead associated with storing millions of individual files.

485 **3. API Specification**

486 **3.1. 1:1 Verification**

487 **3.1.1. Overview**

488 The 1:1 testing will proceed in three phases: preparation of enrollment templates; preparation of verification templates;
489 and matching. These are detailed in Table 19.

490 **Table 19 – Functional summary of the 1:1 application**

Phase	Description	Performance Metrics to be reported by NIST
Initialization	Function to read configuration data, if any.	None
Enrollment	Given $K \geq 1$ input images of an individual, the implementation will create a proprietary enrollment template. NIST will manage storage of these templates.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template
Verification	Given $K \geq 1$ input images of an individual, the implementation will create a proprietary verification template. NIST will manage storage of these templates.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template.
Matching (i.e. comparison)	Given a proprietary enrollment and a proprietary verification template, compare them to produce a similarity score.	Statistics of the time taken to compare two templates. Accuracy measures, primarily reported as DETs.

491
492 NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process
493 invocations, or a mixture of both.

494 **3.1.2. API**

495 **3.1.2.1. Initialization**

496 The NIST test harness will call the initialization function in Table 20 before calling template generation or matching.

497 **Table 20 – Initialization**

Prototype	ReturnCode initializeVerification(const std::string &configurationLocation);	
		Input
Description	This function initializes the SDK under test. It will be called by the NIST application before any call to the Table 21 functions <code>convertMultifaceToEnrollmentTemplate</code> or <code>convertMultifaceToVerificationTemplate</code> . The implementation under test should set all parameters.	
Input Parameters	configurationLocation	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST, not hardwired by the provider. The names of the files in this directory are hardwired in the implementation and are unrestricted.
Output Parameters	none	
Return Value	See Table 17 for all valid return code values.	

498 **3.1.2.2. Template generation**

499 The functions of Table 21 support role-specific generation of template data. Template format is entirely proprietary.

500 **Table 21 – Template generation**

Prototypes	ReturnCode convertMultifaceToEnrollmentTemplate(const Multiface &inputFaces,	
		Input

	PersonRep &template);	Output
	int32_t convertMultifaceToVerificationTemplate(const Multiface &inputFaces, PersonRep &template);	Input
		Output
Description	Takes a <code>Multiface</code> and populates a <code>PersonRep</code> object. In all cases, even when unable to extract features, the template generated for the <code>PersonRep</code> should be a template that may be passed to the <code>matchTemplates</code> function without error. That is, this routine must internally encode "template creation failed" and the matcher must transparently handle this.	
Input Parameters	inputFaces	Implementations must alter their behavior according to the number of images contained in the structure, and the types per Table 10.
Output Parameters	template	A <code>PersonRep</code> object that represents a single template generated from the <code>Multiface</code> .
Return Value	See Table 17 for all valid return code values.	

501 **3.1.2.3. Matching**

502 Matching of one enrollment against one verification template shall be implemented by the function of Table 22.

503 **Table 22 – Template matching**

Prototype	ReturnCode matchTemplates(const uint8_t *verificationTemplate, const uint32_t verificationTemplateSize, const uint8_t *enrollmentTemplate, const uint32_t enrollmentTemplateSize, double &similarity);	Input
		Output
Description	Compare two proprietary templates and output a similarity score, which need not satisfy the metric properties. When either or both of the input templates are the result of a failed template generation (see Table 21), the similarity score shall be -1 and the function return value shall be <code>VerifTemplateError</code> .	
Input Parameters	verificationTemplate	A template generated from a call to <code>convertMultifaceToVerificationTemplate()</code> .
	verificationTemplateSize	The size, in bytes, of the input verification template $0 \leq N \leq 2^{32} - 1$
	enrollmentTemplate	A template generated from a call to <code>convertMultifaceToEnrollmentTemplate()</code> .
	enrollmentTemplateSize	The size, in bytes, of the input enrollment template $0 \leq N \leq 2^{32} - 1$
Output Parameters	similarity	A similarity score resulting from comparison of the templates, on the range <code>[0,DBL_MAX]</code> . See section 0.
Return Value	See Table 17 for all valid return code values.	

504

505 **3.2. 1:N Identification**

506 **3.2.1. Overview**

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

509 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

510 **Table 23 – Procedural overview of the identification test**

Phase	#	Name	Description	Performance Metrics to be reported by NIST
Enrollment	E1	Initialization	<p>Give the implementation advance notice of the number of individuals and images that will be enrolled.</p> <p>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.</p> <p>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.</p> <p>After enrollment, NIST may rename and relocate the enrollment directory - the implementation should not depend on the name of the enrollment directory.</p>	
	E2	Parallel Enrollment	<p>For each of N individuals, pass multiple images of the individual to the implementation for conversion to a combined template. The implementation will return a template to the calling application.</p> <p>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.</p> <p>Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers. The same person will not be enrolled twice.</p>	<p>Statistics of the times needed to enroll an individual.</p> <p>Statistics of the sizes of created templates.</p> <p>The incidence of failed template creations.</p>
	E3	Finalization	<p>Permanently finalize the enrollment directory. This supports, for example, adaptation of the image-processing functions, adaptation of the representation, writing of a manifest, indexing, and computation of statistical information over the enrollment dataset.</p> <p>The implementation is permitted read-write-delete access to the enrollment directory during this phase.</p>	<p>Size of the enrollment database as a function of population size N and the number of images.</p> <p>Duration of this operation. The time needed to execute this function shall be reported with the preceding enrollment times.</p>

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.	Statistics of the time needed for this operation.
	S2	Template preparation	For each probe, create a template from a set of input images. 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.
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 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.

511 **3.2.2. API**

512 **3.2.2.1. Initialization of the enrollment session**

513 Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization function of Table
514 24.

515 **Table 24 – Enrollment initialization**

Prototype	ReturnCode initializeEnrollmentSession(const std::string &configurationLocation, const std::string &enrollmentDirectory, const uint32_t numPersons, const uint32_t numImages);	
		Input
		Input
		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 <code>convertMultifaceToEnrollmentTemplate</code> . 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 <code>enrollmentDirectory</code> .	
Input Parameters	configurationLocation	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollmentDirectory	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 $0 \leq N \leq 2^{32} - 1$ (e.g. 1 million)
	numImages	The total number of images that will be enrolled, summed over all identities $0 \leq M \leq 2^{32} - 1$ (e.g. 1.8 million)
Output Parameters	none	
Return Value	See Table 17 for all valid return code values.	

516 **3.2.2.2. Enrollment**

517 A `Multiface` is converted to one or more enrollment templates (based on the number of persons found in the
518 `Multiface`) using the function of Table 25.

519 **Table 25 – Enrollment feature extraction**

Prototypes	ReturnCode convertMultifaceToEnrollmentTemplates(const Multiface &inputFaces, std::vector<PersonRep> &templates);	
		Input
		Output
Description	<p>This function takes a <code>Multiface</code> and outputs a vector of <code>PersonRep</code> objects. If the function executes correctly (i.e. returns a successful 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 <code>Multiface</code> in which no persons appear, a valid output is an empty vector (i.e. <code>size () == 0</code>).</p> <p>If the function gives a non-zero exit status:</p> <ul style="list-style-type: none"> – 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. <p>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 of section 3.2.2.3.</p>	
Input Parameters	inputFaces	An instance of a Table 9 structure. Implementations must alter their behavior according to the number of images contained in the structure.
Output Parameters	templates	For each person detected in the <code>Multiface</code> , the function shall identify the person's estimated eye centers for each images/video frame where the person's eye coordinates can be calculated. The eye coordinates shall be captured in the <code>PersonRep. eyeCoordinates</code> variable, which is a vector of <code>EyePair</code> objects. For videos, the frame number from the video of where the eye coordinates were detected shall be captured in the <code>EyePair. frameNum</code> 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 <code>EyePair</code> for those frames.
Return Value	See Table 17 for all valid return code values.	

520 **3.2.2.3. Finalize enrollment**

521 After all templates have been created, the function of Table 26 will be called. This freezes the enrollment data. After this
522 call the enrollment dataset will be forever read-only.

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

526 **Implementations shall not *move* the input data. Implementations shall not point to the input data. Implementations
527 should not assume the input data would be readable after the call. Implementations must, at a minimum, copy the
528 input data or otherwise extract what is needed for search.**

529 **Table 26 – Enrollment finalization**

Prototypes	ReturnCode finalizeEnrollment (const std::string &enrollmentDirectory, const std::string &edbName, const std::string &edbManifestName);	
		Input
		Input
		Input
Description	<p>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 2.4. The enrollment directory permissions will be read + write.</p> <p>The function supports post-enrollment, developer-optional, bookkeeping operations, including indexing, tree-building, statistical processing and data re-ordering for fast in-memory searching. The function will generally be called in a</p>	

	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	enrollmentDirectory	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 2.4. 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. This is a NIST-provided input – implementers shall not internally hard-code or assume any values.
	edbManifestName	The name of a single file containing the EDB manifest of section 2.4. The file may be opened directly. It is not necessary to prepend a directory name. This is a NIST-provided input – implementers shall not internally hard-code or assume any values.
Output Parameters	None	
Return Value	See Table 17 for all valid return code values.	

530 **3.2.2.4. Pre-search feature extraction**

531 **3.2.2.5. Initialization**

532 Before `Multifaces` are sent to the identification feature extraction function, the test harness will call the initialization
 533 function in Table 27.

534 **Table 27 – Identification feature extraction initialization**

Prototype	ReturnCode initializeFeatureExtractionSession(const std::string &configurationLocation, const std::string &enrollmentDirectory);	
		Input
		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 $M \geq 1$ calls to <code>convertMultifaceToIdentificationTemplate</code> . Caution: The implementation should tolerate execution of $P > 1$ processes on the one or more machines each of which may be reading from this same enrollment directory in parallel. The implementation has read-only access to its prior enrollment data.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollment_directory	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	
Return Value	See Table 17 for all valid return code values.	

535 **3.2.2.6. Feature extraction**

536 A `Multiface` is converted to an atomic identification template using the function of Table 28. The result may be stored
 537 by NIST, or used immediately. The SDK shall not attempt to store any data.

538 **Table 28 – Identification feature extraction**

Prototypes	ReturnCode convertMultifaceToIdentificationTemplates(const Multiface &inputFaces, std::vector<PersonRep> &templates);	
		Input
		Output
Description	This function takes a <code>Multiface</code> as input and populates a vector of <code>PersonRep</code> with the number of persons detected from the <code>Multiface</code> . The implementation could call <code>vector::push_back</code> 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	inputFaces	One or more faces, or a video clip.
Output Parameters	templates	For each person detected in the video, the function shall create a <code>PersonRep</code> object, populate it with a template and eye coordinates for each image or video frame where eyes were detected, and add it to the vector.
Return Value	See Table 17 for all valid return code values.	

539 **3.2.2.7. Initialization**

540 The function of Table 29 will be called once prior to one or more calls of the searching function of Table 30. The function
541 might set static internal variables so that the enrollment database is available to the subsequent identification searches.

542 **Table 29 – Identification initialization**

Prototype	ReturnCode initializeIdentificationSession(const std::string &configurationLocation, const std::string &enrollmentDirectory);	
		Input Input
Description	This function reads whatever content is present in the <code>enrollmentDirectory</code> , for example a manifest placed there by the <code>finalizeEnrollment</code> function.	
Input Parameters	configurationLocation	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollmentDirectory	The read-only top-level directory in which enrollment data was placed.
Return Value	See Table 17 for all valid return code values.	

543 **3.2.2.8. Search**

544 The function of Table 30 compares a proprietary identification template against the enrollment data and returns a
545 candidate list.

546 **Table 30 – Identification search**

Prototype	ReturnCode identifyTemplate(const PersonRep &idTemplate, const uint32_t candidateListLength, std::vector<Candidate> &candidateList);	
		Input Input Output
Description	This function searches an identification template against the enrollment set, and outputs a vector containing <code>candidateListLength</code> Candidates. Each candidate shall be populated by the implementation and added to <code>candidateList</code> . Note that <code>candidateList</code> 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 <code>convertMultifaceToIdentificationTemplates()</code> - If the value returned by that function was non-zero the contents of <code>idTemplate</code> will not be used and this function (i.e. <code>identifyTemplate</code>) will not be called.
	candidateListLength	The number of candidates the search should return.
Output Parameters	candidateList	A vector containing <code>candidateListLength</code> objects of candidates. The datatype is defined in Table 16 . Each candidate shall be populated by the implementation. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.
Return Value	See Table 17 for all valid return code values.	

547 NOTE: Ordinarily the calling application will set the input candidate list length to operationally typical values, say $0 \leq L \leq$
548 200, and $L \ll N$. However, there is interest in the presence of mates much further down the candidate list. We may
549 therefore extend the candidate list length such that L approaches N . We may measure the dependence of search
550 duration on L .

551 **3.3. Face Detection**

552 **3.3.1. API**

553 **3.3.1.1. Initialization**

554 Before any calls to `detectFaces` are made, the NIST test harness will make a call to the initialization of the function in
 555 Table 31.

556 **Table 31 – SDK initialization**

Prototype	ReturnCode initializeDetection(const std::string &configurationLocation);	
		Input
Description	This function initializes the SDK under test. It will be called by the NIST application before any call to the function <code>detectFaces</code> . The SDK under test should set all parameters.	
Input Parameters	configurationLocation	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST. It is not hardwired by the provider. The names of the files in this directory are hardwired in the SDK and are unrestricted.
Output Parameters	none	
Return Value	See Table 17 for all valid return code values.	

557 **3.3.1.2. Face detection**

558 The function of Table 32 supports the detection of faces in an image. An image may contain one or more faces.

559 **Table 32 – Face detection**

Prototypes	ReturnCode detectFaces(const Image &inputImage, std::vector<BoundingBox> &boundingBoxes);	
		Input
		Output
Description	This function takes an Image as input, and populates a vector of <code>BoundingBox</code> with the number of faces detected from the input image. The implementation could call <code>vector::push_back</code> to insert into the vector.	
Input Parameters	inputImage	An instance of a struct representing a single image from Table 8.
Output Parameters	boundingBoxes	For each face detected in the image, the function shall create a <code>BoundingBox</code> (see Table 13), populate the x, y, width, height of the bounding box, and add it to the vector.
Return Value	See Table 17 for all valid return code values.	

560

561 **3.4. Clustering**

562 **3.4.1. Definitions**

563 Clustering is the act of grouping imagery of the same individuals. If a large image collection has N images in which $P \geq 0$
 564 subjects appear, an implementation should return N lists. The n -th list contains zero or more hypotheses about who
 565 appears in the n -th input image. Each hypothesis is comprised of: a bounding box; an integer subject identifier; and a
 566 similarity score. A similarity is a measure of confidence that a hypothesized identity truly shares the same face as others
 567 in that cluster. Subject identifiers are labels assigned by the algorithm.

568 Clustering will, in general, produce detection errors (where a person is not found at all), and both false positive and
 569 negative associations where, respectively, multiple persons appear in one cluster, one person exists in several clusters. A
 570 single image can contain one or more faces in it.

571 **3.4.2. API**

572 **3.4.2.1. Initialization**

573 Before any calls to `clusterIdentities` are made, the NIST test harness will make a call to the initialization of the
 574 function in Table 33.

575 **Table 33 – SDK initialization**

Prototype	ReturnCode initializeClustering(const std::string &configurationLocation);		Input
Description	This function initializes the SDK under test. It will be called by the NIST application before any call to the function <code>clusterIdentities</code> . The SDK under test should set all parameters.		
Input Parameters	configurationLocation	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST. It is not hardwired by the provider. The names of the files in this directory are hardwired in the SDK and are unrestricted.	
Output Parameters	none		
Return Value	See Table 17 for all valid return code values.		

576 **3.4.2.2. Cluster Identities**

577 The implementation shall implement the function given in Table 34.

578 **Table 34 – Clustering**

Prototype	ReturnCode clusterIdentities(const std::vector<Image> &inputFaces, std::vector<ClusterMembersInImage> &assignments);		Input	Output
Description	This function compares a template against the enrollment set, and outputs a list of cluster hypotheses. NIST will pre-allocate the <code>assignments</code> vector with <code>input_faces.size()</code> objects before the call. It is up to the implementations to populate the <code>ClusterMembersInImage.members</code> vector based on the number of faces found in the images. Implementations should, for each <code>Image</code> , <code>input_faces[i]</code> , assign the hypothesized cluster assignment(s) in <code>assignments[i]</code> .			
Input Parameters	inputFaces	N Images from $P \geq 0$ subjects. There are $n_i \geq 1$ images from individual i .		
Output Parameters	assignments	N lists of cluster assignments.		
Return Value	See Table 17 for all valid return code values.			

579
 580 Question to reviewers: Are there use cases for face clustering where the number of subjects/clusters is a known priori,
 581 and should NIST evaluate this scenario?

582 **4. References**

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 2012	Patrick Grother and Mei Ngan, Face Recognition Vendor Test (FRVT) Performance of Face Identification Algorithms, NIST Interagency Report 8009, May 26, 2014.
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.
IJBA	See for example the images here: http://www.nist.gov/itl/iad/ig/facechallenges.cfm As documented here: Klare et al. Pushing the Frontiers of Unconstrained Face Detection and Recognition: IARPA Janus Benchmark A , CVPR, June 2015.
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
MEDS	NIST Special Database 32, Volume 1 and Volume 2 are available at: http://www.nist.gov/itl/iad/ig/sd32.cfm . MEDS-II is an update to MEDS-I and was published in February 2011. Note that NIST does not provide "training" data per se - this differs from the paradigm often used in academic research where a model is trained, tested and validated. Instead CHEXIA-FACE follows operational reality: software is typically shipped "as is" with a fixed internal representation that is designed to be usable "off the shelf" without training and with only minimal configuration.
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

583

584
585

Annex A

Submission of Implementations to the CHEXIA-FACE

586 A.1 Submission of implementations to NIST

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

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

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

595 A.2 How to participate

596 Those wishing to participate in CHEXIA-FACE testing must do all of the following, on the schedule listed on Page 2.

597 — IMPORTANT: Follow the instructions for cryptographic protection of your SDK and data here.

598 <http://www.nist.gov/itl/iad/ig/encrypt.cfm>

599 — Send a signed and fully completed copy of the *Application to Participate in the Child Exploitation Image Analytics –*
600 *Face Recognition Evaluation (CHEXIA-FACE)*. This is available at <http://www.nist.gov/itl/iad/ig/chexia-face.cfm>. This
601 must identify, and include signatures from, the Responsible Parties as defined in the application. The properly signed
602 CHEXIA-FACE Application to Participate shall be sent to NIST as a PDF.

603 — Provide an SDK (Software Development Kit) library which complies with the API (Application Programmer Interface)
604 specified in this document.

605 • Encrypted data and SDKs below 20MB can be emailed to NIST at chexia.face@nist.gov.

606 • Encrypted data and SDKS above 20MB shall be

607 EITHER

608 ▪ Split into sections AFTER the encryption step. Use the unix "split" commands to make 9MB chunks,
609 and then rename to include the filename extension need for passage through the NIST firewall.

610 ▪ `you% split -a 3 -d -b 9000000 libCHEXIAFACE_enron_A_02.tgz.gpg`

611 ▪ `you% ls -l x??? | xargs -iQ mv Q libCHEXIAFACE6_enron_A_02_Q.tgz.gpg`

612 ▪ Email each part in a separate email. Upon receipt NIST will

613 ▪ `nist% cat chexiaface_enron_A02_*.tgz.gpg >`
614 `libCHEXIAFACE_enron_A_02.tgz.gpg`

615 OR

616 ▪ Made available as a file.zip.gpg or file.zip.asc download from a generic http webserver⁸,

617 OR

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

CHEXIA-FACE 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.

619 A.3 Implementation validation

620 Registered Participants will be provided with a small validation dataset and test program available on the website

621 <http://www.nist.gov/itl/iad/ig/chexia-face.cfm> shortly after the final evaluation plan is released.

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

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

625 Software submitted shall implement the CHEXIA-FACE API Specification as detailed in the body of this document.

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

629

630

631

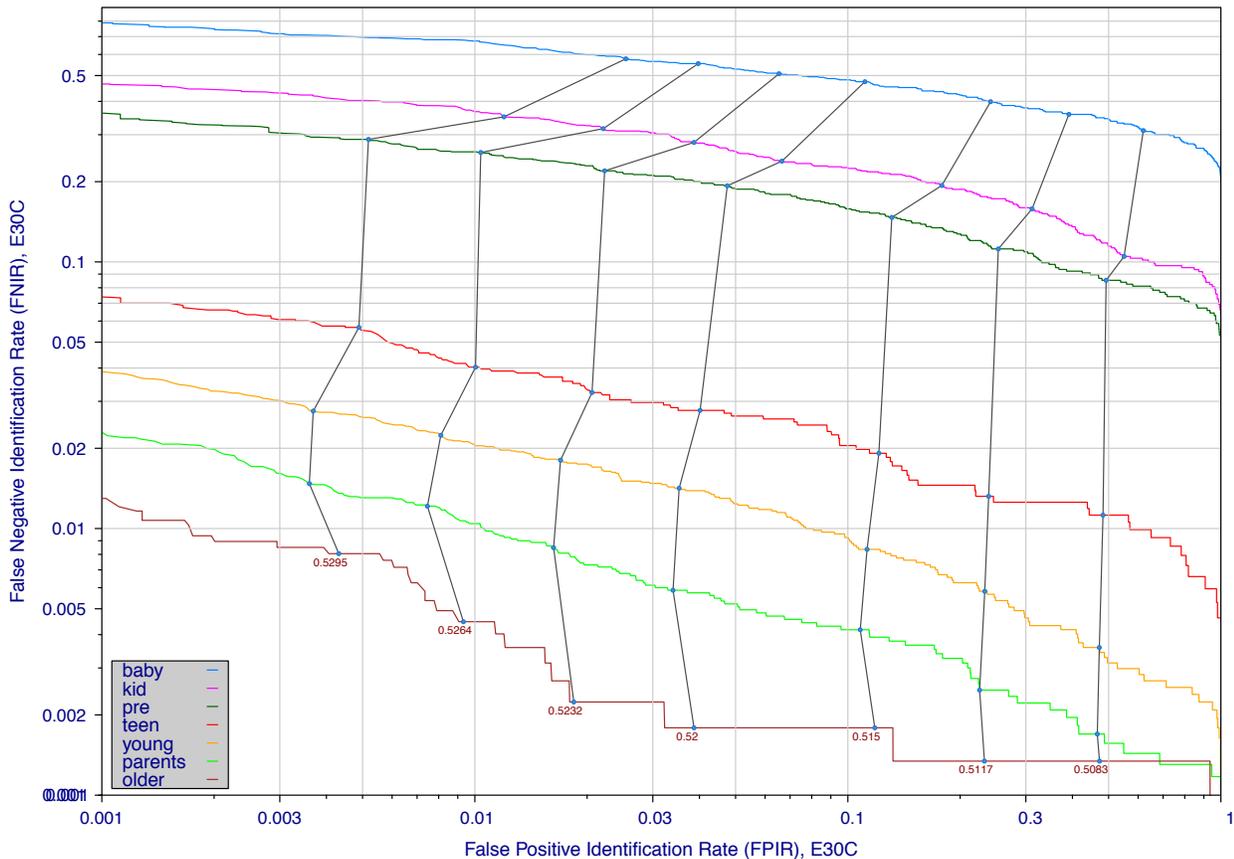
Annex B Effect of Age on Face Identification Accuracy

632 For the most accurate algorithm provided to NIST’s FRVT evaluation in late 2013 the Figure below shows the one-to-many
 633 identification accuracy for subjects from particular age groups. The images are visa images. We enrolled a first image
 634 from each of $N = 19972$ individuals. Thereafter, we executed one mated search from those individuals to allow estimation
 635 of False Negative Identification Rate (FNIR, aka “miss rate”). We also executed 203,082 non-mated searches to allow
 636 computation of the False Positive Identification Rate (FPIR, aka “false alarm rate”).

637 Results for 40 algorithms appear in Annex A of NIST Interagency Report 8009⁹. The discussion from that report is:

- 638 — **Recognition is progressively easier with advancing age:** All algorithms exhibit a strong dependence of FNIR on age. This effect is
 639 very large, spanning a factor of ten from infant to senior, and a factor of around five from teen to senior. Miss rates for *older*
 640 persons are very low: at a fixed FPIR of 0.005, the most accurate algorithm, E30C, gives FNIR of 0.008 for persons over age 55,
 641 0.027 for *young* 20-somethings, and 0.057 for teenagers. For younger persons, the miss rates climb rapidly to 0.29 for *pre-teens*,
 642 0.4 for *kids*, to 0.7 for *babies*. This progression is common to all algorithms.
- 643 — **Young children are more difficult to recognize:** Identification miss rates (FNIR) ascend rapidly for *pre-teens*, *kids* and the youngest
 644 individuals. For the *baby* group, 0 to about 3 years old, identification fails more often that it succeeds, i.e. FNIR is above 50%.
 645 While the sample size is small (57 subjects), error rates are so high that the result remains significant. This result applies for image
 646 pairs collected on average 1.6 years apart (Table 13) and will be in considerable part due to the craniofacial shape change
 647 associated with rapid growth. The extent to which smooth “feature-less” skin texture affects FNIR is unknown. Likewise the pose
 648 variations inherent in photographing children have not been quantified.
- 649 — **Young children are more difficult to discriminate:** All of the algorithms exhibit higher false positive identification rates for younger
 650 subjects. The grey lines in Figure 11, which link points of equal threshold, slope upwards to the right, indicating simultaneously
 651 that younger subjects are less easy to recognize as themselves but also less easy to tell apart. This indicates that younger
 652 individuals are more difficult to discriminate from other individuals.

653



654

⁹ http://biometrics.nist.gov/cs_links/face/frvt/frvt2013/NIST_8009.pdf

