Face Recognition Vendor Test
Ongoing

Still Face 1:N Identification
Application Programming Interface
VERSION 2.0

Patrick Grother
Mei Ngan
Kayee Hanaoka
Information Access Division
Information Technology Laboratory

February 14, 2022

NIST

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

Revision History

1

Date	Version	Description
April 1, 2019	1.0	Initial document
September 9, 2020	1.0.1	- Update link to General Evaluation Specifications document - Adjust the legal similarity score range
August 16, 2021	1.0.2	Removed FRVT 1:1 pre-requisite. Developers may now participate in FRVT 1:N without having to participate in FRVT 1:1
November 3, 2021	1.0.3	- Added clarification that multi-threading is allowed in the finalizeEnrollment() function - Removed holdover text from 2018 - Added clarification on function time limits to be based on a single core
January 7, 2022	2.0	Add second version of createTemplate() function from Section 9.4.4 that supports the existence of multiple people in an image

2
3

Table of Contents

List of Tables

1. FRVT 1:N

1.1. Scope

This document establishes a concept of operations and an application programming interface (API) for evaluation of one-to-many face recognition algorithms applied to faces appearing in 2D still photographs. The test will include cooperative portrait images, e.g. mugshots[1] as well as non-cooperative images[2]. The API communicates the type of image to the algorithm, so that feature extraction may be tailored.

2. Rules for participation

2.1. Relation to the FRVT Ongoing 1:1 Evaluation

Since February 2017, NIST has been running an ongoing evaluation of one-to-one face verification algorithms, FRVT 1:1. This allows any developer to submit algorithms at any time, once every few months, thereby better aligning development and evaluation schedules. It will benefit developers to submit their core 1:1 recognition algorithms to the FRVT 1:1 process.

~~IMPORTANT: Participation in the FRVT Ongoing 1:1 is **required** for participation in FRVT Ongoing 1:N. Participants must meet minimum accuracy requirements in the 1:1 test in order to submit to the 1:N.~~

3. General FRVT Evaluation Specifications

General and common information shared between all Ongoing FRVT tracks are documented in the FRVT General Evaluation Specifications document - https://pages.nist.gov/frvt/api/FRVT_common.pdf. This includes rules for participation, hardware and operating system environment, software requirements, reporting, and common data structures that support the APIs.

4. Core accuracy metrics

This test will execute open-universe searches. That is, some proportion of searches will not have an enrolled mate. From the candidate lists returned by algorithms, NIST will compute and report accuracy metrics, primarily:

— False negative identification rate (FNIR) – the proportion of mated searches which do not yield a mate within the top R ranks and at or above threshold, T.

— False positive identification rate (FPIR) – the proportion of non-mated searches returning any (1 or more) candidates at or above a threshold, T.

— Selectivity – the number of non-mated candidates returned at or above a threshold, T. This quantity has a value running from 0 to L, the number of candidates requested. It may be fractional, as it is estimated as a count divided by the number of non-mate searches.

These quantities are estimated from candidate lists produced by requesting the top L most similar candidates to the search. We do not intend to execute searches requesting only those candidates above a specified input threshold.

We will report FNIR, FPIR and selectivity by sweeping the threshold over the interval [0, infinity). Error tradeoff plots (FNIR vs. FPIR, parametric on threshold) will be the primary reporting mechanism.

We will also report FNIR by sweeping a rank R over the interval [1, L] to produce (the complement of) the cumulative match characteristic (CMC).

We will report proportions of attempted template generations that fail to produce a viable template – i.e failure to enroll rate (FTE).

---

[1] The MEDS database includes sample mugshots.
[2] The IJB-C database includes non-cooperative, in-the-wild images.

76 ## 5. Application relevance

77 NIST anticipates reporting FNIR in two FPIR / Selectivity regimes:

78 — Investigation mode: Given candidate lists and a threshold of zero, the CMC metric is relevant to investigational
79 applications where human examiners will adjudicate candidates in decreasing order of similarity. This is common in
80 law enforcement "lead generation".

81 — Identification mode: We will apply (high) thresholds to candidate lists and report FNIR values relevant to
82 identification applications where human labor is matched to the tolerable number of false positives per unit time.
83 This is used in duplicate-ID detection searches for credential issuance and, more so, in surveillance applications.

84 Developers are encouraged to submit variants tailored to minimize FNIR in the two FPIR regimes, and to explore the
85 speed-accuracy trade space.

86 ## 6. Limits

87 ### 6.1. Time limits

88 The elemental functions of the implementations shall execute under the time constraints of Table 1. These time limits
89 apply to the function call invocations defined in section 9. Assuming the times are random variables, NIST cannot regulate
90 the maximum value, so the time limits are median values. This means that the median of all operations should take less
91 than the identified duration. Timing will be estimated from at least 1000 separate invocations of each elemental function.
92 Timing will be measured on Xeon R CPU E5-2630 v4 @ 2.20GHz processors.

93 **Table 1 – Processing time limits in seconds, per 640 x 480 color image**

Function	1:N
Template Generation: One image to one template	1 sec (1 core)
1:N finalization (on gallery of 1 million enrolled templates)	40000 sec

1:N search for:
    — N = 1 million enrolled templates
    — L = 100 returned candidates
	10 sec (1 core)

94 ### 6.2. Template size limits

95 NIST anticipates evaluating performance with N well in excess of 10^7. For implementations that represent a gallery in
96 memory with a linear data structure, the memory of our machines implies a limit on template sizes. Thus, for a template
97 size B, the total memory requirement would be about NB. NIST anticipates running the largest N values on machines
98 equipped with 768GB of memory. With N = 25 million, templates should not exceed 32KB.

99 The API, however, supports multi-stage searches and read access of the disk during the 1:N search. Disk access would
100 likely be very slow. In all cases, algorithms shall conform to the search duration limits given in Table 1, with linear scaling.

101 ## 7. Implementation Library Filename

102 The core library shall be named as libfrvt_1N_<*provider*>_<*sequence*>.so, with
103 • provider: single word, non-infringing name of the main provider. Example: acme
104 • sequence: a three digit decimal identifier to start at 000 and incremented by 1 every time a library is sent to
105 NIST. Example: 007
106
107 Example core library names: *libfrvt_1N_acme_000.so, libfrvt_1N_mycompany_006.so.*
108 Important: Public results will be attributed with the provider name and the 3-digit sequence number in the submitted
109 library name.

110 8. Data structures supporting the API

111 The general data structures supporting this API are documented in the FRVT - General Evaluation Specifications document
112 available at https://pages.nist.gov/frvt/api/FRVT_common.pdf. The data structures specific to this particular test are
113 described within this document. The header files are published at https://github.com/usnistgov/frvt.

114 8.1. File structure for enrolled template collection

115 To support this 1:N test, NIST will concatenate enrollment templates into a single large file, the EDB (for enrollment
116 database). The EDB is a simple binary concatenation of proprietary templates. There is no header. There are no
117 delimiters. The EDB may be many gigabytes in length.

118 This file will be accompanied by a manifest; this is an ASCII text file documenting the contents of the EDB. The manifest
119 has the format shown as an example in Table 2. If the EDB contains N templates, the manifest will contain N lines. The
120 fields are space (ASCII decimal 32) delimited. There are three fields. Strictly speaking, the third column is redundant.

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

123 **Table 2 – Enrollment dataset template manifest**

Field name	Template ID	Template Length	Position of first byte in EDB
Datatype required	std::string	uint64_t	uint64_t
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

124
125 The EDB scheme avoids the file system overhead associated with storing millions of small individual files.

126 8.1.1. Gallery Type
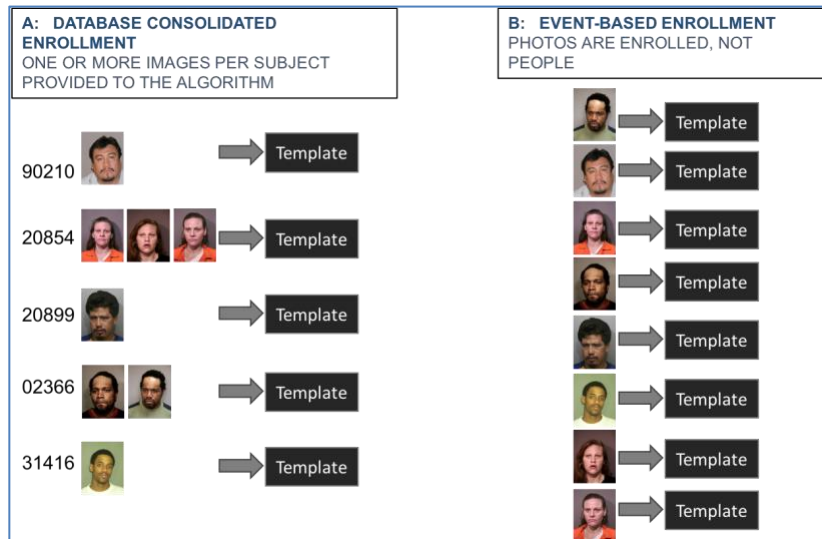
127
128 **Figure 1 – Illustration of consolidated versus unconsolidated enrollment database[3]**

129 Figure 1 illustrates two types of galleries:

---

[3] The face images contained in this figure are from the publicly available Special Database 32 - Multiple Encounter Dataset (MEDS). https://www.nist.gov/itl/iad/image-group/special-database-32-multiple-encounter-dataset-meds

130 — **Consolidated:** The database is formed by enrolling all images of a subject under a common identity label. The result
131   is a gallery with N identities and N templates. This type of gallery presents us with the cleanest experimental design,
132   "one needle in a haystack" scenario. It allows algorithms to perform image and feature level fusion. Operationally it
133   requires high integrity biographical information to maintain.

134 — **Unconsolidated:** The database is formed by enrolling photographs without regard to whether the subject already has
135   already been enrolled or not. Under this scheme, different images of the same person can exist in the gallery under
136   different subject identifiers, that is, there are N identities, and M > N database entries.

137 During gallery finalization, algorithms will be provided with an enumerated label from Table 3 which specifies the type of
138 gallery being processed.

139   **Table 3 – Labels describing gallery composition**

Label as C++ enumeration	Meaning
`enum class GalleryType {`	
`    Consolidated,`	Consolidated, subject-based enrollment
`    Unconsolidated`	Unconsolidated, event-based or photo-based enrollment
`};`	

140 **8.1.2.   Data structure for result of an identification search**

141 All identification searches shall return a candidate list of a NIST-specified length. The list shall be sorted with the most
142 similar matching entries list first with lowest rank. The data structure shall be that of Table 4.

143   **Table 4 – Structure for a candidate**

	C++ code fragment	Remarks
1.	`typedef struct Candidate`	
2.	`{`	
3.	`    bool isAssigned;`	If the candidate computation succeeded, this value is set to true. False otherwise. If value is set to false, similarityScore and templateId will be ignored entirely.
4.	`    std::string templateId;`	The Template ID from the enrollment database manifest defined in clause 8.1.
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 false-negative and false-positive identification rates. The similarity score values should be reported on the range that is used in the developer's software products. Larger values indicate more likelihood that the two samples are from the same person. However, we require scores to be non-negative. Developers often use [0,1], for example. Our test reports include various plots with threshold values e.g. FMR(T), to allow end-users to set thresholds in operations. These plots may become difficult to interpret if scores span many orders of magnitude.
6.	`} Candidate;`	

144

9. API specification

146 FRVT 1:N participants shall implement the relevant C++ prototyped interfaces of section 9. C++ was chosen in order to
147 make use of some object-oriented features.

148

149 Please note that included with the FRVT 1:N validation package (available at https://github.com/usnistgov/frvt) is a "null"
150 implementation of this API. The null implementation has no real functionality but demonstrates mechanically how one
151 could go about implementing this API.

9.1.   Header File

153 The prototypes from this document will be written to a file named **frvt1N.h** and will be available to implementers at
154 https://github.com/usnistgov/frvt.

9.2.   Namespace

156 All supporting data structures will be declared in the `FRVT` namespace. All API interfaces/function calls for this track will
157 be declared in the `FRVT_1N` namespace.

9.3.   Overview

159 The 1:N identification application proceeds in three phases: enrollment, finalization and identification. The identification
160 phase includes separate probe feature extraction and search stages.

161 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
- − support measurement of template insertion and removal times into an enrollment database

162   **Table 5 – Procedural overview of the 1:N test**

Phase	#	Name	Description	Performance Metrics to be reported by NIST
Enrollment	E1	Initialization	**initializeTemplateCreation(TemplateRole=Enrollment_1N)** 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-only** access to the configuration directory.	
	E2	Parallel Enrollment	**createTemplate(TemplateRole=Enrollment_1N)** For each of N individuals, pass K >= 1 images of the individual to the implementation for conversion to a template. The implementation will return a template to the calling application. 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.	Statistics of the times needed to enroll an individual. Statistics of the sizes of created templates. The incidence of failed template creations.
Gallery Finalization	F1	Finalization	**finalizeEnrollment()** 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. The implementation is permitted **read-write-delete access** to the enrollment directory and **read-only access** to the configuration directory during this phase. **Note:** finalizeEnrollment() will be called in a separate process than the enrollment functions.	Size of the enrollment database as a function of population size N. Duration of this operation. The time needed to execute this function shall be reported with the preceding enrollment times.

Probe Template Creation	S1	Initialization	**initializeTemplateCreation(TemplateRole=Search_1N)** 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-only** access to the configuration directory.	Statistics of the time needed for this operation.
	S2	Template preparation	**createTemplate(TemplateRole=Search_1N)** For each probe, create a template from K >= 1 images. The result of this step is a search template. Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers.	Statistics of the time needed for this operation. Statistics of the size of the search template.
Search	S3	Initialization	**initializeIdentification()** Tell the implementation the location of an enrollment directory that contains the gallery files produced from the **finalize()** function. The enrollment directory will always contain a successfully finalized gallery (i.e. will never be empty). 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. **Note:** The search functions (initializeIdentification(), identifyTemplate()) will be called in a separate process from the enrollment functions, therefore, you <u>cannot</u> assume that initializeTemplateCreation() is called by the test harness prior to the search functions.	Statistics of the time needed for this operation.
	S4	Search	**identifyTemplate()** A template is searched against the enrollment database. Developers shall not attempt to improve the duration of the identifyTemplate() function by offloading any of its processing into the createTemplate() function.	Statistics of the time needed for this operation. Accuracy metrics - Type I + II error rates. Failure rates.
Gallery Insert	G1	Initialization	**initializeIdentification()** 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 duration of the operation.
	G2	Template insertion into gallery	**galleryInsertID()** galleryInsertID() is executed one or more times to insert a template created with createTemplate(TemplateRole=Enrollment_1N) into the gallery.	Statistics of the duration of the operation.
	G3	Search	**identifyTemplate()** A template is searched against the enrollment database.	Statistics of the duration of the operation. Accuracy metrics - Type I + II error rates.

9.4.   API

9.4.1.   Interface

The software under test must implement the interface `Interface` by subclassing this class and implementing each method specified therein.

	C++ code fragment	Remarks
1.	Class Interface	
2.	{ public:	
3.	static std::shared_ptr<Interface> getImplementation();	Factory method to return a managed pointer to the `Interface` object. This function is implemented by the submitted library and must return a managed pointer to the `Interface` object.
4.	// Other functions to implement	
5.	};	

167 There is one class (static) method declared in `Interface.getImplementation()` which must also be
168 implemented. This method returns a shared pointer to the object of the interface type, an instantiation of the
169 implementation class. A typical implementation of this method is also shown below as an example.

C++ code fragment	Remarks

```#include "frvt1N.h"```
 	
 ```using namespace FRVT_1N;```
 	
 ```NullImpl:: NullImpl () { }```
 	
 ```NullImpl::~ NullImpl () { }```
 	
 ```std::shared_ptr<Interface>```
 ```Interface::getImplementation()```
 ```{```
 ```   return std::make_shared<NullImpl>();```
 ```}```
 ```// Other implemented functions```	

9.4.2. Initialization of template creation

171 Before any feature extraction/template creation calls are made, the NIST test harness will call the initialization function of
172 Table 6. This function will be called BEFORE any calls to fork() are made.

173 **Table 6 – Template creation initialization**

Prototype	ReturnStatus initializeTemplateCreation(	
	const std::string &configDir,	Input
	TemplateRole role);	Input
Description	This function initializes the implementation under test and sets all needed parameters in preparation for template creation. This function will be called N=1 times by the NIST application, prior to parallelizing M >= 1 calls to createTemplate() via fork(). This function will be called from a single process/thread.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	role	A value from the TemplateRole enumeration that indicates the intended usage of the template to be generated. In this case, either Enrollment_1N or Search_1N.
Output Parameters	None	
Return Value	See General Evaluation Specifications document for all valid return code values.	

9.4.3. Template Creation from one or more images of exactly one person

175 The function of Table 7 supports role-specific generation of template data from one or more images of exactly one
176 person. A Multiface is converted to a single template using this function.

177 Some of the proposed datasets include K > 2 images per person for some persons. This affords the possibility to model a
178 recognition scenario in which a new image of a person is compared against all prior images. Use of multiple images per
179 person has been shown to elevate accuracy over a single image.

180 For this test, NIST will enroll K >= 1 images under each identity. Normally the probe will consist of a single image, but NIST
181 may examine the case that it could consist of multiple images. Ordinarily, the probe images will be captured after the
182 enrolled images of a person. The method by which the face recognition implementation exploits multiple images is not
183 regulated. The test seeks to evaluate developer provided technology for multi-presentation fusion.

184 This document defines a template to be the result of applying feature extraction to a set of K >= 1 images. An algorithm
185 might internally fuse K feature sets into a single model or maintain them separately - in any case the resulting proprietary
186 template is contained in a contiguous block of data. All identification functions operate on such multi-image templates.

187 **Table 7 – Template Creation/Feature Extraction from one or more images of exactly one person**

Prototypes	ReturnStatus createTemplate(	
	const Multiface &faces,	Input
	TemplateRole role,	Input
	std::vector<uint8_t> &templ,	Output
	std::vector<EyePair> &eyeCoordinates);	Output
Description	Takes a Multiface and outputs a proprietary template and associated eye coordinates. The vector to store the template will be initially empty, and it is up to the implementation to populate it with the appropriate data.	
	For enrollment templates (TemplateRole=Enrollment_1N): If the function executes correctly (i.e. returns a successful return code), the template will be enrolled into a gallery. The NIST calling application may store the resulting template, concatenate many templates, and pass the result to the enrollment finalization function (see section 9.4.4). The resulting template may also be inserted immediately into previously finalized gallery. When the implementation fails to produce a template (i.e. returns a non-successful return code), it shall still return a blank template (which can be zero bytes in length). The template will be included in the enrollment database/manifest like all other enrollment templates, but is not expected to contain any feature information.	
	IMPORTANT: NIST's application writes the template to disk. Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function of section 9.4.4.	
	For identification/probe templates (TemplateRole=Search_1N): 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-successful return status, the output template will not be used in subsequent search operations.	
Input Parameters	face	Input Multiface
	role	Label describing the type/role of the template to be generated. In this case, it will either be Enrollment_1N or Search_1N.
Output Parameters	templ	The output template. The format is entirely unregulated. This will be an empty vector when passed into the function, and the implementation can resize and populate it with the appropriate data.
	eyeCoordinates	The function shall return the estimated eye centers for the input face image.
Return Value	See General Evaluation Specifications document for all valid return code values.	

188 **9.4.4.   Template Creation of one or more people detected from an image**

189 This function supports role-specific generation of one or more templates that correspond to one or more people detected
190 in an image. Some of the proposed test images include K > 1 persons for some images and situations where the subject of
191 interest may or may not be the foreground face (largest face in the image). This function allows the implementation to
192 return a template for each person detected in the image. For testing, NIST will

193   1.   Enroll one more templates from a single call to this function or the function of Table 7

194   2.   Generate one or more search templates from a single call to this function or the function of Table 7

195   3.   Search all templates generated from 2) against the enrollment database

196 4. Use the **maximum** similarity score or best rank across all searches from 3) in our calculation of FNIR and FPIR
197 (this applies to both genuine and imposter searches)

198 **NOTE:** The implementation must be able to match any combination of enrollment and search templates generated from
199 this function and the function of Table 7. In other words, the output template format should be consistent between this
200 function and the function of Table 7.

201

202 **Table 8 – Template Creation/Feature Extraction of one or more people detected from an image**

Prototypes	ReturnStatus createTemplate(	
	const Image &image,	Input
	TemplateRole role,	Input
	std::vector<std::vector<uint8_t>> &templs,	Output
	std::vector<EyePair> &eyeCoordinates);	Output
Description	This function supports template generation of one or more people detected from a single image. It takes a single input image and outputs one or more proprietary templates and associated eye coordinates based on the number of people detected. The vectors to store the template(s) and eye coordinates will be initially empty, and it is up to the implementation to populate them with the appropriate data.	
	For enrollment templates (TemplateRole=Enrollment_1N): If the function executes correctly (i.e. returns a successful return code), the template(s) will be enrolled into a gallery. The NIST calling application may store the resulting template(s), concatenate many templates, and pass the result to the enrollment finalization function (see section 9.4.4). The resulting template(s) may also be inserted immediately into previously finalized gallery. When the implementation fails to produce a template (i.e. returns a non-successful return code), it shall still return a blank template (which can be zero bytes in length). The template will be included in the enrollment database/manifest like all other enrollment templates, but is not expected to contain any feature information.	
	IMPORTANT: NIST's application writes the template to disk. Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function of section 9.4.4.	
	For identification/probe templates (TemplateRole=Search_1N): The NIST calling application may commit the template(s) to permanent storage, or may keep it only in memory (the developer implementation does not need to know). If the function returns a non-successful return status, the output template(s) will not be used in subsequent search operations.	
Input Parameters	image	A single image that contains one or more people in the photo
	role	Label describing the type/role of the template to be generated. In this case, it will either be Enrollment_1N or Search_1N.
Output Parameters	templs	A vector of output template(s). The format of the template(s) is entirely unregulated. This will be an empty vector when passed into the function, and the implementation can resize and populate it with the appropriate data.
	eyeCoordinates	For each person detected in the image, the function shall return the estimated eye centers. This will be an empty vector when passed into the function, and the implementation shall populate it with the appropriate number of entries. Values in eyeCoordinates[i] shall correspond to templs[i].
Return Value	See General Evaluation Specifications document for all valid return code values.	

203

204 **9.4.5. Finalization**

205 After all templates have been created, the function of Table 9 will be called. This freezes the enrollment data. After this
206 call the enrollment dataset will be forever read-only.

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

210   **Implementations shall not move the input data.   Implementations shall not point to the input data. Implementations**
211   **should not assume the input data will be readable after the call. Implementations must, at a minimum, copy the input**
212   **data or otherwise extract what is needed for search.**

213   **Table 9 – Enrollment finalization**

Prototypes	ReturnStatus finalizeEnrollment(	
	const std::string &configDir,	Input
	const std::string &enrollmentDir,	Input
	const std::string &edbName,	Input
	const std::string &edbManifestName,	Input
	GalleryType galleryType);	Input
Description	This function takes the name of the top-level directory where the enrollment database (EDB) and its manifest have been stored. These are described in section 8.1. The enrollment directory permissions will be read + write.	
	The function supports post-enrollment, developer-optional, book-keeping operations, statistical processing and data re-ordering for fast in-memory searching. 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.	
	This function will be called from a single process/thread. Implementation of this function does not need to be single-threaded (i.e., developers may use multiple threads within this function).	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollmentDir	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 8.1. While the file will have read-write-delete permission, the implementation 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 8.1. 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.
	galleryType	A label from Table 3 specifying the composition of the gallery.
Output Parameters	None	
Return Value	See General Evaluation Specifications document for all valid return code values.	

214   ### 9.4.6.   Search Initialization

215   The function of Table 10 will be called once prior to one or more calls of the searching function of Table 11 and the gallery
216   insert and delete functions of Section 9.4.8. The function might set static internal variables so that the enrollment
217   database is available to the subsequent identification searches. This function will be called BEFORE any calls to fork() are
218   made.

219   **Table 10 – Identification initialization**

Prototype	ReturnStatus initializeIdentification(	
	const string &configDir,	Input
	const string &enrollmentDir);	Input
Description	This function reads whatever content is present in the enrollmentDir, for example a manifest placed there by the finalizeEnrollment() function. This function will be called from a single process/thread.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.

	enrollmentDir	The read-only top-level directory in which enrollment data was placed. This directory will contain the gallery files produced from the **finalize()** function. The enrollment directory will always contain a successfully finalized gallery (i.e. will never be empty).
Return Value	See General Evaluation Specifications document for all valid return code values.	

220 ### 9.4.7. Search

221 The function of Table 11 compares a proprietary identification template against the enrollment data and returns a
222 candidate list.

223 **Table 11 – Identification search**

Prototype	ReturnStatus identifyTemplate (	
	const std::vector<uint8_t> &idTemplate,	Input
	const uint32_t candidateListLength,	Input
	std::vector<Candidate> &candidateList,	Output
	bool &decision);	Output
Description	This function searches a template against the enrollment set, and outputs a list of candidates. The candidateList vector will initially be empty, and the implementation shall populate the vector with candidateListLength entries.	
Input Parameters	idTemplate	A template from createTemplate(TemplateRole=Search_1N) - If the value returned by that function was non-zero the contents of idTemplate will not be used and this function (i.e. identifyTemplate) will not be called.
	candidateListLength	The number of candidates the search should return
Output Parameters	candidateList	A vector containing "candidateListLength " objects of candidates. The datatype is defined in section 8.1.1. 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.
	decision	A best guess at whether there is a mate within the enrollment database. If there was a mate found, this value should be set to true, Otherwise, false. Many such decisions allow a single point to be plotted alongside a DET.
Return Value	See General Evaluation Specifications document for all valid return code values.	

224

225 NOTE: Ordinarily the calling application will set the input candidate list length to operationally typical values, say $0 \leq L \leq$
226 200, and L << N. We will measure the dependence of search duration on L.

227 ### 9.4.8. Gallery insertion of templates

228 The functions of this section insert a new template into a finalized gallery.

229 **Table 12 – Insertion of template into a gallery**

Prototype	ReturnStatus galleryInsertID(	
	const std::vector<uint8_t> &templ,	Input
	const std::string &id);	Input
Description	This function inserts a template with an associated id into an existing finalized gallery. Invocation of this function will always be preceded by a call to initializeIdentification(), which will provide the location of the finalized gallery to be loaded into memory. One or more calls to identifyTemplate() may be made after calling this function.	
	The template ID will not exist in the database already, so a 1:N duplicate search is not necessary.	
	This function will be called from a single process/thread.	
Input Parameters	templ	A template created via createTemplate(TemplateRole=Enrollment_1N)
	id	An identifier associated with the enrollment template
Return Value	See General Evaluation Specifications document for all valid return code values.	

230