Face Recognition Vendor Test
Ongoing

Still Face and Iris 1:1 Verification
Application Programming Interface (API)
VERSION 6.0

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

Contact via frvt@nist.gov

April 6, 2023

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

Revision History

Date	Version	Description
April 1, 2019	4.0	Initial document
June 24, 2020	4.0.1	Update feature extraction times in Table 1.3 from 1000ms to 1500ms
September 9, 2020	4.0.2	Update link to General Evaluation Specifications document Adjust the legal similarity score range
March 22, 2021	4.0.3	Update 1:1 matching time limit in Table 1.3 from 5 milliseconds to 0.1 milliseconds (or 100 microseconds)
January 7, 2022	5.0	Add second version of createTemplate() function in Section 4.4.4 that supports the existence of multiple people in an image
February 2, 2022	5.0.1	Add Figure 2 and Table 3 to illustrate the second version of createTemplate() function from Section 4.4.4
March 24, 2022	5.0.2	Add verbiage to be more explicit about algorithmic behavior when the software fails to find a face in an image in Sections 4.4.3 and 4.4.4
April 6, 2023	6.0	Add support for iris recognition Remove references to deprecated Multiface data structure

Table of Contents

1. FRVT 1:1 3
1.1. Scope 3
1.2. General FRVT Evaluation Specifications 3
1.3. Time limits 3
2. Data structures supporting the API 3
3. Implementation Library Filename 3
4. API Specification 3
4.1. Header File 4
4.2. Namespace 4
4.3. Overview 4
4.4. API 5

List of Tables

Table 1 – Processing time limits in milliseconds, per 640 x 480 image 3
Table 2 – Functional summary of the 1:1 application of Figure 1 4
Table 3 – Functional summary of the 1:1 application of Figure 2 5
Table 4 – Initialization 6
Table 5 – Template generation from one or more images of exactly one person 7
Table 6 – Template generation of one more people detected from a single image 8
Table 7 – Template matching 9

List of Figures

Figure 1 – Schematic of 1:1 verification (template generation of one or more images of exactly one person) 4
Figure 2 – Schematic of 1:1 verification (template generation of one or more people detected in an image) 5

1. FRVT 1:1

1.1. Scope

This document establishes a concept of operations and an application programming interface (API) for evaluation of face recognition (FR) or iris recognition implementations submitted to NIST's ongoing Face Recognition Vendor Test. This API is for the 1:1 identity verification track. Separate API documents will be published for future additional tracks to FRVT.

1.2. 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.

1.3. Time limits

The elemental functions of the implementations shall execute under the time constraints of Table 1. These time limits apply to the function call invocations defined in section 3. Assuming the times are random variables, NIST cannot regulate the maximum value, so the time limits are median values. This means that the median of all operations should take less than the identified duration.

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

NOTE: For developers that cannot meet the required time limit for matching two templates, please contact frvt@nist.gov.

Table 1 – Processing time limits in milliseconds, per 640 x 480 image

Function	1:1 verification
Feature extraction enrollment	1500 (1 core) 640x480 pixels
Feature extraction for verification	1500 (1 core) 640x480 pixels
Matching	0.1 (1 core)

2. Data structures supporting the API

The data structures supporting this API are documented in the FRVT - General Evaluation Specifications document available at https://pages.nist.gov/frvt/api/FRVT_common.pdf with corresponding header file named *frvt_structs.h* published at https://github.com/usnistgov/frvt.

3. Implementation Library Filename

The core library shall be named as libfrvt_11_<***provider***>_<***sequence***>.so, with

- provider: single word, non-infringing name of the main provider. Example: acme
- sequence: a three digit decimal identifier to start at 000 and incremented by 1 every time a library is sent to NIST. Example: 007

Example core library names: *libfrvt_11_acme_000.so, libfrvt_11_mycompany_006.so.*
Important: Public results will be attributed with the provider name and the 3-digit sequence number in the submitted library name.

4. API Specification

FRVT 1:1 participants shall implement the relevant C++ prototyped interfaces in Section 4.4. C++ was chosen in order to make use of some object-oriented features.

4.1. Header File

The prototypes from this document will be written to a file named **frvt11.h** and will be available to implementers at https://github.com/usnistgov/frvt.

4.2. Namespace

All supporting data structures will be declared in the `FRVT` namespace. All API interfaces/function calls for this track will be declared in the `FRVT_11` namespace.

4.3. Overview

The 1:1 testing will proceed in the following phases: optional offline training; preparation of enrollment templates; preparation of verification templates; and matching. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.

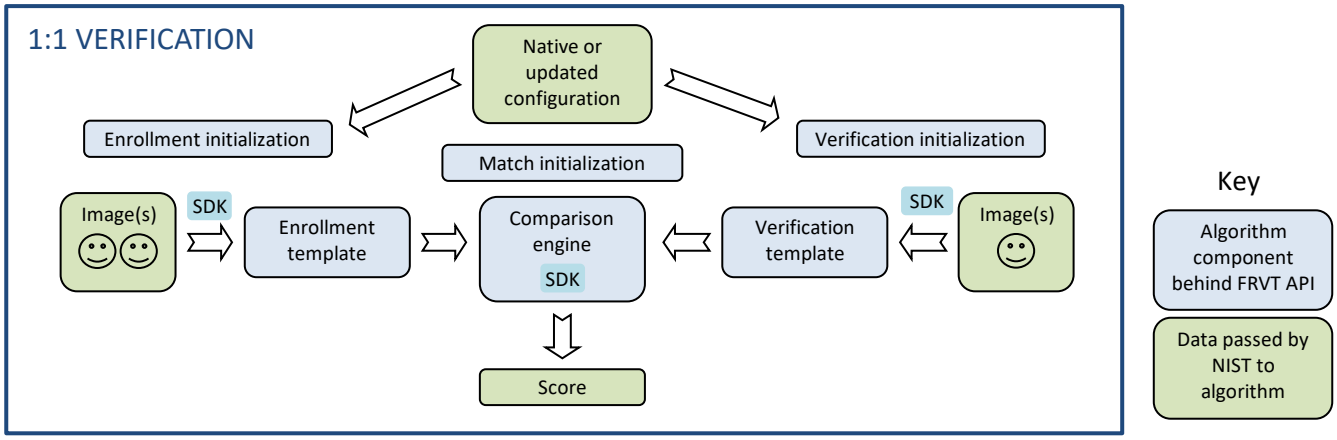

Figure 1 – Schematic of 1:1 verification (template generation of one or more images of exactly one person)

Table 2 – Functional summary of the 1:1 application of Figure 1

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. That is, createTemplate(role=FRVT::TemplateRole::Enrollment_11) will be called. 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. That is, createTemplate(role=FRVT::TemplateRole::Verification_11) will be called. 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, including for partitions of the input datasets.

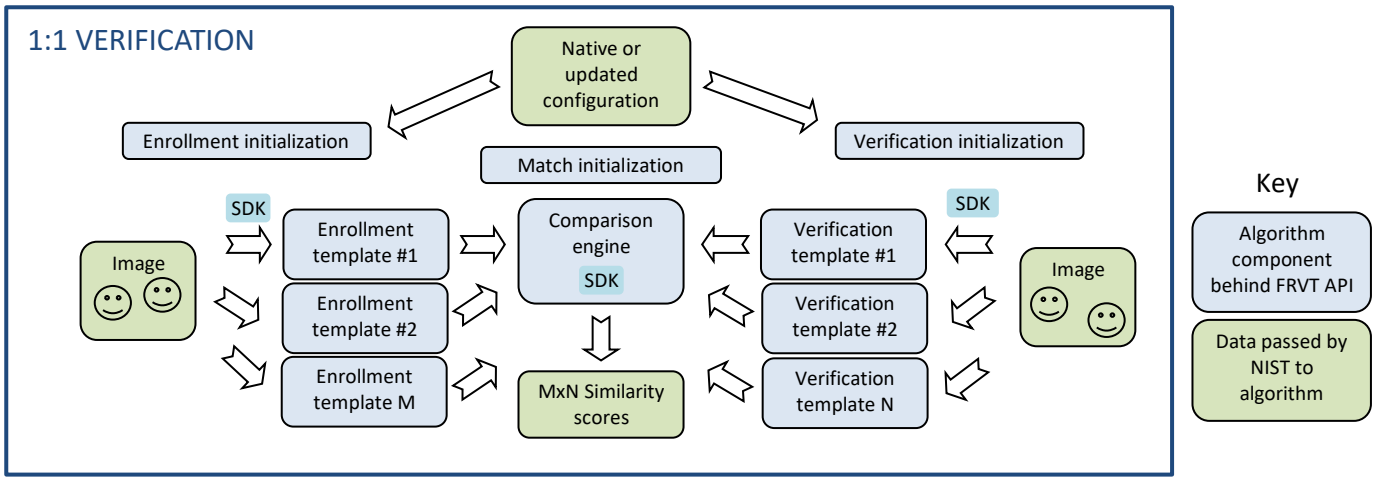

Figure 2 – Schematic of 1:1 verification (template generation of one or more people detected in an image)

Table 3 – Functional summary of the 1:1 application of Figure 2

Phase	Description	Performance Metrics to be reported by NIST
Initialization	Function to read configuration data, if any.	None
Enrollment	Given K = 1 input image, the implementation will create M proprietary enrollment templates based on the number of people detected in the image. That is, createTemplate(role=FRVT::TemplateRole::Enrollment_11) will be called. NIST will manage storage of these templates.	Statistics of the time needed to produce M templates. Statistics of template size. Rate of failure to produce a template.
Verification	Given K = 1 input image, the implementation will create N proprietary verification templates based on the number of people detected in the image. That is, createTemplate(role=FRVT::TemplateRole::Verification_11) will be called. NIST will manage storage of these templates.	Statistics of the time needed to produce N templates. Statistics of template size. Rate of failure to produce a template.
Matching (i.e. comparison)	Given a M proprietary enrollment templates and N proprietary verification templates, cross compare them to produce MxN similarity scores.	Statistics of the time taken to compare two templates. Accuracy measures, primarily reported as DETs, including for partitions of the input datasets.

4.4. API

4.4.1. 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.	`virtual ReturnStatus initialize(` `const std::string &configDir ) = 0;`	Supports algorithm initialization

4.	`virtual ReturnStatus createFaceTemplate(`
	`const std::vector<Image> &faces,`
	`TemplateRole role,`
	`std::vector<uint8_t> &templ,`
	`std::vector<EyePair> &eyeCoordinates) = 0;`
		Supports template generation from one or more face images of exactly one person

5.	`virtual ReturnStatus createIrisTemplate(`
	`  const std::vector<Image> &irises,`
	`  TemplateRole role,`
	`  std::vector<uint8_t> &templ,`
	`  std::vector<IrisAnnulus> &irisLocations) = 0;`
		Supports template generation from one or more iris images of exactly one person

6.	`virtual ReturnStatus createFaceTemplate(`
	`  const Image &image,`
	`  TemplateRole role,`
	`  std::vector<std::vector<uint8_t>> &templs,`
	`  std::vector<EyePair> &eyeCoordinates) = 0;`
		Supports template generation of one or more people detected from a single image

7.	`virtual ReturnStatus matchTemplates(`
	`  const std::vector<uint8_t> &verifTemplate,`
	`  const std::vector<uint8_t> &enrollTemplate,`
	`  double &similarity) = 0;`
		Supports comparison between two templates

8.	`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.
9.	`};`	

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

	C++ code fragment	Remarks

	`#include "frvt11.h"`
	
	`using namespace FRVT_11;`
	
	`NullImpl:: NullImpl () { }`
	
	`NullImpl::~ NullImpl () { }`
	
	`std::shared_ptr<Interface>`
	`Interface::getImplementation()`
	`{`
	`  return std::make_shared<NullImpl>();`
	`}`
	
	`// Other implemented functions`	

4.4.2. Initialization

The NIST test harness will call the initialization function in Table 4 before calling template generation or matching. This function will be called BEFORE any calls to `fork()`[1] are made.

Table 4 – Initialization

Prototype	ReturnStatus initialize(const string &configDir);	 Input
Description	This function initializes the implementation under test. It will be called by the NIST application before any call to `createTemplate()` or `matchTemplates()`. The implementation under test should set all parameters. This function will be called N=1 times by the NIST application, prior to parallelizing M >= 1 calls to `createTemplate()` via `fork()`.	
Input Parameters	configDir	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 General Evaluation Specifications document for all valid return code values.	

[1] http://man7.org/linux/man-pages/man2/fork.2.html

4.4.3. Template generation from one or more images of exactly one person

The functions of Table 5 supports role-specific generation of template data from one or more images of exactly one person. Template format is entirely proprietary. Some of the proposed datasets include K > 2 image per person for some persons. This affords the possibility to model a recognition scenario in which a new image of a person is compared against all prior images. Use of multiple images per person has been shown to elevate accuracy over a single image.

NOTE: For any given submission, developers may only implement ONE of the functions in Table 5. That is, a single submission may only support face recognition or iris recognition. For the functions that are not implemented, the function shall return ReturnCode::NotImplemented.

Using this function, NIST will enroll K >= 1 images under each identity. The method by which the face or iris recognition implementation exploits multiple images is not regulated. The test seeks to evaluate developer provided technology for multi-presentation fusion.

This document defines a template to be a developer defined data-structure, capable of holding zero or one embeddings. A developer may include embeddings, other information derived from the image, a header; the contents is developer defined. An algorithm might internally fuse K feature sets into a single model or maintain them separately. In any case, the resulting proprietary template is contained in a contiguous block of data. **A template may have length zero bytes. In all cases, the matchTemplates() function must accept two templates. The matchTemplates() function will be called even if the developer's implementation of a template is of zero bytes.**

Table 5 – Template generation from one or more images of exactly one person

<table>
<tr><td>Prototype for face recognition</td><td colspan="2">ReturnStatus createFaceTemplate(
const std::vector<Image> &faces,
TemplateRole role,
std::vector<uint8_t> &templ,
std::vector<EyePair> &eyeCoordinates);</td><td>
Input
Input
Output
Output</td></tr>
<tr><td>Prototype for iris recognition</td><td colspan="2">ReturnStatus createIrisTemplate(
const std::vector<Image> &irises,
TemplateRole role,
std::vector<uint8_t> &templ,
std::vector<IrisAnnulus> &irisLocations);</td><td>
Input
Input
Output
Output</td></tr>
<tr><td></td><td colspan="3"></td></tr>
<tr><td>Description</td><td colspan="3">Takes a vector of image(s) and outputs a proprietary template and associated eye coordinates or irisLocations. The vectors to store the template and eye coordinates/iris locations will be initially empty, and it is up to the implementation to populate them with the appropriate data. In all cases, even when unable to extract features, the output shall be a template that may be passed to the matchTemplates() function without error. That is, this routine must internally encode "template creation failed", and the matcher must transparently handle this. The table below specifies algorithmic behavior based on whether a face or iris was detected/features were extracted from the input image(s).
<table>
<tr><th></th><th>ReturnCode</th><th>Output Template</th></tr>
<tr><td>Zero faces/irises detected</td><td>A non-successful return code</td><td>One template (could be zero bytes)</td></tr>
<tr><td>K = 1 faces/irises detected</td><td>Success</td><td>One template</td></tr>
</table></td></tr>
<tr><td rowspan="2">Input Parameters</td><td>faces or irises</td><td colspan="2">Implementations must alter their behavior according to the number of images contained in the structure and the TemplateRole type.</td></tr>
<tr><td>role</td><td colspan="2">Label describing the type/role of the template to be generated. Valid values are FRVT::TemplateRole::Enrollment_11 or FRVT::TemplateRole::Verification_11.</td></tr>
<tr><td rowspan="2">Output Parameters</td><td>templ</td><td colspan="2">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.</td></tr>
<tr><td>eyeCoordinates or irisLocations</td><td colspan="2">The function shall return
– For face images, eye coordinates – the estimated eye centers for left and right eyes
– For iris images, iris locations - estimates of the limbus center and pupil and limbus radii</td></tr>
</table>

Return Value	See General Evaluation Specifications document for all valid return code values.

4.4.4. Template generation of one or more people detected from an image

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

1. Generate one or more enrollment templates from a single call to this function or the function of Table 5
2. Generate one or more verification templates from a single call to this function or the function of Table 5
3. Match all enrollment templates from 1) with all verification templates from 2)
4. Use the **maximum** similarity score across all template comparisons from 3) in our calculation of FMR and FNMR (this applies to both genuine and imposter comparisons)

NOTE: The implementation must be able to match any combination of enrollment and verification templates generated from this function and the function of Table 5. In other words, the output template format should be consistent between this function and the function of Table 5. **A template may have length zero bytes. In all cases, the matchTemplates() function must accept two templates. The matchTemplates() function will be called even if the developer's implementation of a template is of zero bytes.**

Table 6 – Template generation of one more people detected from a single image

<table>
<tr><td>Prototypes</td><td colspan="2">ReturnStatus createFaceTemplate(</td><td></td></tr>
<tr><td></td><td colspan="2">const Image &image,</td><td>Input</td></tr>
<tr><td></td><td colspan="2">TemplateRole role,</td><td>Input</td></tr>
<tr><td></td><td colspan="2">std::vector<<std::vector<uint8_t>> &templs,</td><td>Output</td></tr>
<tr><td></td><td colspan="2">std::vector<EyePair> &eyeCoordinates);</td><td>Output</td></tr>
<tr><td>Description</td><td colspan="3">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. If the implementation is unable to extract features, the output shall still contain a single template that may be passed to the matchTemplates() function without error. That is, this routine must internally encode "template creation failed", and the matcher must transparently handle this. The table below specifies algorithmic behavior based on the number of faces detected from the input image.
<table>
<tr><th></th><th>ReturnCode</th><th>Output Template</th></tr>
<tr><td>Zero faces detected</td><td>A non-successful return code</td><td>Vector of templates, length = 1</td></tr>
<tr><td>K >= 1 faces detected</td><td>Success</td><td>Vector of templates, length = K</td></tr>
</table></td></tr>
<tr><td rowspan="2">Input Parameters</td><td>image</td><td colspan="2">A single image that contains one or more people in the photo</td></tr>
<tr><td>role</td><td colspan="2">Label describing the type/role of the template to be generated. Valid values are FRVT::TemplateRole::Enrollment_11 or FRVT::TemplateRole::Verification_11.</td></tr>
<tr><td rowspan="2">Output Parameters</td><td>templs</td><td colspan="2">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.</td></tr>
<tr><td>eyeCoordinates</td><td colspan="2">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].</td></tr>
<tr><td>Return Value</td><td colspan="3">See General Evaluation Specifications document for all valid return code values.</td></tr>
</table>

137

138 4.4.5. Matching

139 Matching of one enrollment against one verification template shall be implemented by the function of Table 7.

140 **Table 7 – Template matching**

Prototype	ReturnStatus matchTemplates(	
	const std::vector<uint8_t> &verifTemplate,	Input
	const std::vector<uint8_t> &enrollTemplate,	Input
	double &score);	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 5), the similarity score shall be -1 and the function return value shall be `VerifTemplateError`.	
Input Parameters	verifTemplate	A verification template from createTemplate(role=Verification_11). The underlying data can be accessed via verifTemplate.data(). The size, in bytes, of the template could be retrieved as verifTemplate.size().
	enrollTemplate	An enrollment template from createTemplate(role=Enrollment_11). The underlying data can be accessed via enrollTemplate.data(). The size, in bytes, of the template could be retrieved as enrollTemplate.size().
Output Parameters	score	A score resulting from comparison of the templates. Measure of similarity or dissimilarity between the enrollment template and verification template. – For face recognition, a similarity score - higher is more similar – For iris recognition, a non-negative measure of dissimilarity (maybe a distance) - lower is more similar 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 rates. The score values should be reported on the range that is used in the developer's software products. 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.
Return Value	See General Evaluation Specifications document for all valid return code values.	

141