Face Recognition Vendor Test
Ongoing

Face Recognition Quality Assessment
Application Programming Interface (API)

VERSION 1.0

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

Contact via frvt@nist.gov

April 23, 2019

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

Table of Contents

List of Tables

1. FRVT Quality

1.1. Scope

This document establishes an application programming interface (API) for evaluation of face recognition (FR) implementations submitted to NIST's Ongoing Face Recognition Vendor Test (FRVT) Face Recognition Quality Assessment (FRQA) track. Separate API documents are/will be published for current and 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://www.nist.gov/system/files/documents/2019/03/20/frvt_common_1.0.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 90-th percentiles. This means that 90% of all operations should take less than the identified duration.

The time limits apply per image.

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

Function	
scalarQuality()	5000 (1 core)

2. Data structures supporting the API

The data structures supporting this API are documented in the FRVT - General Evaluation Specifications document, 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_quality_<***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_quality_acme_000.so, libfrvt_quality_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 Quality participants shall implement the relevant C++ prototyped interfaces in Section 4.3 . 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 **frvt_quality.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_QUALITY` namespace.

4.3. API

4.3.1. 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;`	
4.	`virtual ReturnStatus scalarQuality( const Image &face, double &quality) = 0;`	
5.	`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.
6.	`};`	

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

4.3.2. Initialization

The NIST test harness will call the initialization function in Table 2 before calling any of the quality assessment functions of this API. This function will be called BEFORE any calls to `fork()`[1] are made.

Table 2 – 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 calls the quality assessment functions of this API. The implementation under test should set all parameters. This function	

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

	will be called N=1 times by the NIST application, prior to parallelizing M >= 1 calls to any other functions 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.	

4.3.3. Scalar Quality Assessment from a Single Image

The functions of Table 3 supports quality assessment of a single face image. Here, quality scores should represent predictors of recognition accuracy. The default use-case is during enrollment – checking that an image is suitable to become the reference in an authoritative database. A second use-case is quality being used *during* a verification or identification transaction to select the image most likely to match the reference image. The reference image is assumed to be unavailable for matching (e.g. because it is on a remote server). In both cases, the quality algorithm should express whether the input would match a canonical frontal portrait image (i.e. one that conforms to the ISO/ICAO standard).

Table 3 – Quality scalar from a single image

Prototypes	ReturnStatus scalarQuality(	
	const Image &face,	Input
	double &quality);	Output
Description	This function takes an image and outputs a quality scalar. The algorithm will be supplied with a label describing the type of image via Image::Label, and it is up to the implementation to alter its behavior based on the image type (e.g., ISO (full-frontal) versus Wild (off-angle).	
Input Parameters	face	Single face image
Output Parameters	quality	For each image in the faces vector, an assessment of image quality as described below: scalarQuality(): overall quality assessment Legal quality values are • [0,100] - The value should have a monotonic decreasing relationship with false non-match rate anticipated for this sample if it was compared with a pristine image of the same person. So, a low value indicates high expected FNMR. • A value of -1.0 indicates a failed attempt to calculate a quality score or the value is unassigned.
Return Value	See General Evaluation Specifications document for all valid return code values.	