# Text Recognition Algorithm Independent Test (TRAIT)

An Evaluation Activity under the DHS Science & Technology Child Exploitation Image Analytics Program (CHEXIA)

Homeland Security

Science and Technology

## Concept, Evaluation Plan and API

Version 0.6, October 26, 2015

Patrick Grother, Mei Ngan, Afzal Godil

Contact via trait2016@nist.gov

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

16 **Provisional Timeline of the TRAIT 2016 Evaluation**

| | | |
|---|---|---|
| Phase 0<br>API Development | 2015-10-05 | Draft evaluation plan |
| | 2015-11-15 | Final evaluation plan |
| Phase 1 | 2015-12-01 | Participation starts: Algorithms may be sent to NIST |
| | **2016-02-08** | **Last day for submission of algorithms to Phase 1** |
| | 2016-03-07 | Interim results released to Phase 1 participants |
| Phase 2 | **2016-06-12** | **Last day for submission of algorithms to Phase 2** |
| | 2016-07-12 | Interim results released to Phase 2 participants |
| Phase 3 | **2016-10-19** | **Last day for submission of algorithms to Phase 3** |
| | 2016-Q4 | Release of final public report |

17

18

## Table of Contents



## List of Figures



## List of Tables

# 1. TRAIT

## 1.1. Scope

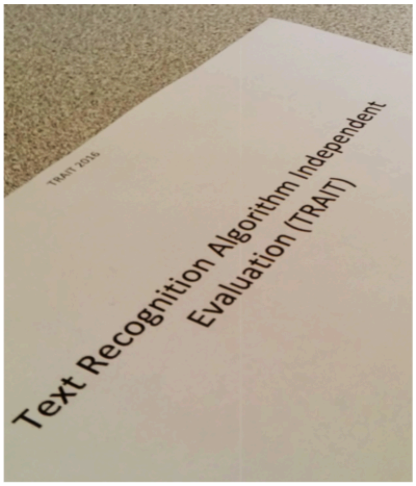
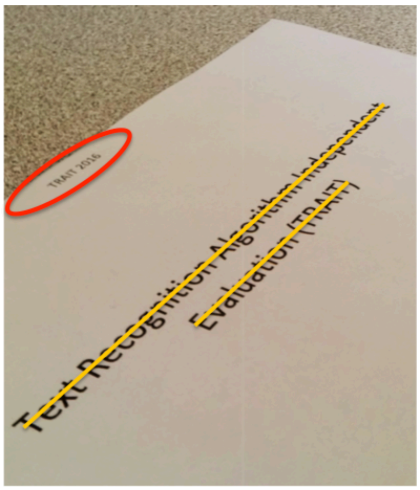
This document establishes a concept of operations and C++ API for evaluation of text-in-image detection and recognition algorithms submitted to NIST's TRAIT program. See http://www.nist.gov/itl/iad/ig/trait-2016.cfm for latest documentation.

TRAIT proceeds as follows. Algorithm developers send compiled C++ libraries to NIST. NIST executes those algorithms on sequestered imagery that has been made available to NIST by, for example, other US Government agencies.

## 1.2. Audience

This document is aimed at universities, commercial entities and other research organizations possessing a capability to detect and recognize unconstrained text in still images. There is no requirement for real-time or streaming-mode processing. An example image appears in Figure 1. It is intended only as an example of out-of-plane text, not as some representation of widely varying test data.

**Figure 1 – Example of inputs and outputs**

| A simple example of out-of-plane text. | Input image showing geometric markup in yellow, and missed detection in red. | Possible Output text and (dummy, nominal) coordinates |
|---|---|---|
|   |  | First string at very top of page is missed<br><br>Text Recognition Algorithm Independent<br>x= (200,580)<br>y = (160,550)<br><br>Evaluation (TRAIT)<br>x = (380,530)<br>y = (320, 500) |

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

NIST is particularly interested to evaluate prototypes that have proven useful in prior evaluations organized underneath the ICDAR conferences (http://2015.icdar.org/program/competitions/) particularly the Robust Reading efforts (http://rrc.cvc.uab.es/)

## 1.3. Market drivers

This test is intended to support a plural marketplace of text recognition systems. Our primary driver is to support forensic investigations of digital media. Specifically, to allow linking of child exploitation events that occur in a common location, or that share other textual clues.

## 1.4. Test data

NIST will run submitted algorithms on several sequestered datasets available to NIST.

The primary dataset is an operational child exploitation collection containing illicit pornographic images. The images are present on digital media seized in criminal investigations. The files include children who range in age from infant through adolescent. Their faces are the subject of a separate face recognition evaluation and development effort (CHEXIA-FACE 2016). Many of the images contain geometrically unconstrained text. This text is human-legible and sometimes has investigational value. Such text is visible on certificates, posters, logos, uniforms, sports apparel, computer screens, business cards, newspapers, books lying on tables, cigarette packets and a long list of more rare objects.

The text is most commonly in English with French, Spanish, German and Cyrillic present in significant quantity. We do not intend to test non-Roman alphabets.

These images are of interest to NIST's partner law enforcement agencies that seek to employ text recognition in investigating this area of serious crime. The primary applications are identification of previously known victims and suspects, as well as detection of new victims and suspects. The presence of text may allow a location to be identified or to generate leads.

## 1.5. Offline testing

TRAIT is intended to mimic operational reality. As an offline test intended to assess the core algorithmic capability of text detection and recognition algorithms, it does not extend to real-time transcription of live image sources. Offline testing is attractive because it allows uniform, fair, repeatable, and efficient evaluation of the underlying technologies. Testing of implementations under a fixed API allows for a detailed set of performance related parameters to be measured.

## 1.6. Phased testing

To support development, TRAIT will be conducted in three phases. In each phase, NIST will evaluate implementations on a first-come-first-served basis and will return results to providers as expeditiously as possible. The final phase will result in public reports. Providers may submit revised SDKs to NIST only after NIST provides results for the prior SDK and invites further submission. The frequency with which a provider may submit SDKs to NIST will depend on the times needed for developer preparation, transmission to NIST, validation, execution and scoring at NIST, and developer review and decision processes.

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

## 1.7. Interim reports

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

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

## 1.8. Final reports

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

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

## 1.9. Application scenarios

The test will include detection and verification tasks for still images. As described in Table 1, the test is intended to support operations in which an automated text recognition engine yields text that can be indexed and retrieved using mainline text retrieval engines.

**Table 1 – Subtests supported under the TRAIT 2016 activity**

| # | | A | B | C |
|---|---|---|---|---|
| 1. | Aspect | Image-to-location | Image-to-text with provided location information | Image-to-text without location information |
| 2. | Languages | Mostly English. Some French, Spanish and German. While some Cyrillic and Chinese appear also, evaluation will be confined to English roman alphabets only. | | |
| 3. | Input | Image | Image and location(s) of text | Image |
| 4. | Output | Given an input image, output detected locations of text. This does not require the algorithm(s) to produce strings of text. | Given an input image and location(s) of text in the image, output strings of text. | Given an input image, output strings of text. This does not require the algorithm to produce the location(s) of text. |

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

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

## 1.10. Options for participation

The following rules apply:

- A participant must properly follow, complete and submit the Annex A Participation Agreement. This must be done once, not before December 1, 2015. It is not necessary to do this for each submitted SDK.
- Participants may submit class C algorithms only if at least 1 class B algorithm is also submitted.
- All submissions shall implement exactly one of the functionalities defined in Table 2. A library shall not implement two or more classes.

**Table 2 – TRAIT 2016 classes of participation**

| Function | Image-to-location | Image-to-text with provided location information | Image-to-text without location information |
|---|---|---|---|
| Class label | A | B | C |
| Must also submit to class | | | B |
| API requirements | 3.1 | 3.2 | 3.3 |

## 1.11. Number and schedule of submissions

The test is conducted in three phases, separated by a few months. The maximum total (i.e. cumulative) number of submissions is regulated in Table 3.

**Table 3 – Cumulative total number of algorithms, by class**

| # | Phase 1 | Total over Phases 1 + 2 | Total over Phases 1 + 2 + 3 |
|---|---|---|---|
| Class A: Image-to-location | 2 | 4 | 6 |
| Class B: Image-to-text with provided location information | 2 | 4 | 6 |
| Class C: Image-to-text without location information | 2 | 4 | 6 |

The numbers above may be increased as resources allow.

NIST cannot conduct surveys over runtime parameters.

## 1.12. Core accuracy metrics

**Recognition:** The evaluation will be performed on the text results provided by each system. We intend to state text recognition accuracy with at least an edit distance such as the Word Error Rate (WER) [1.12a] between the reference text and text provided by the system for each line. WER is calculated with the edit distance with equal cost of deletions, substitutions, and insertions and finally normalize the edit distance by the number of characters in the ground truth words.

[1.12a] J. Fiscus, J. Ajot, N. Radde, and C. Laprun, *Multiple Dimension Levenshtein Edit Distance Calculations for Evaluating Automatic Speech Recognition Systems During Simultaneous Speech*, Proceedings of LREC, 2006.
http://www.itl.nist.gov/iad/mig/publications/storage_paper/lrec06_v0_7.pdf

**Detection:** The text detection task will be evaluated, somewhat similar to prior open evaluations [1.12b]. However, in our case the ground truth text, is defined by line and curve segments instead of bounding boxes. Hence our methodology will use a simple matching distance approach between lines and curves as the criteria.

[1.12b] C. Wolf and J.-M. Jolion. Object count/Area Graphs for the Evaluation of Object Detection and Segmentation Algorithms, International Journal on Document Analysis and Recognition, 8(4):280-296, 2006.
http://liris.cnrs.fr/christian.wolf/software/deteval/index.html

## 1.13. Reporting computational efficiency

NIST will also report timing statistics for all core functions of the submitted SDK implementations.

## 1.14. Hardware specification

NIST intends to execute the software on Dual Intel Xeon E5-2695 3.3 GHz CPUs (14 cores each) with Dual NVIDIA Telsa K40 GPUs. NIST will respond to prospective participants' questions on the hardware by amending this section.

## 1.15. Operating system, compilation, and linking environment

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

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

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

A typical link line might be

```
g++ -I. -Wall -m64 -o trait16test  trait16test.cpp  -L.  –ltrait2016_Enron_A_07
```

The Standard C++ library should be used for development. The prototypes from this document will be written to a file "trait2016.h" which will be included via

```
#include <trait2016.h>
```

The header files will be made available to implementers at http://nigos.nist.gov:8080/trait2016.

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

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

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

## 1.16. Software and Documentation

### 1.16.1. SDK Library and Platform Requirements

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

The core library shall be named according to Table 4. Additional shared object library files may be submitted that support this "core" library file (i.e. the "core" library file may have dependencies implemented in these other libraries).

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

**Table 4 – Implementation library filename convention**

| Form | libTRAIT2016_provider_class_sequence.ending | | | | |
|---|---|---|---|---|---|
| Underscore delimited parts of the filename | libTRAIT2016 | provider | class | sequence | ending |
| Description | First part of the name, required to be this. | Single word name of the main provider EXAMPLE: Enron | Function classes supported in Table 2. 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 | libTRAIT2016_Enron_C_07.so | | | | |

NIST will report the size of the supplied libraries.

### 1.16.2. Configuration and developer-defined data

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

NIST will report the size of the supplied configuration files.

### 1.16.3. Installation and Usage

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

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

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

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

### 1.16.4. Documentation

Participants may provide documentation of the SDK and detail any additional functionality or behavior beyond that specified here. The documentation might include developer-defined error or warning return codes. The documentation shall not include any intellectual property.

## 1.17. Runtime behavior

### 1.17.1. Interactive behavior

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

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

### 1.17.2. Exception Handling

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

### 1.17.3. External communication

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

### 1.17.4. Stateless behavior

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

## 1.18. Threaded computations

All implementations should run without threads, or with exactly one worker thread. This allows NIST to parallelize the test by dividing the workload across many cores and many machines. To expedite testing, for single-threaded libraries, NIST will run up to P = 16 processes concurrently. NIST's calling applications are single-threaded.

## 1.19. Time limits

Given a 12 megapixel input image, the text detection and recognition software should execute in less than 10 seconds.

# 2. Data structures supporting the API

## 2.1. Overview

This section describes separate APIs for the core text detection/recognition applications described in section 1.9. All SDK's submitted to TRAIT 2016 shall implement the functions required by the rules for participation listed before Table 2.

## 2.2. Requirement

TRAIT 2016 participants shall submit an SDK which implements the relevant C++ functions (per class) as specified in Table 2. C++ was chosen in order to make use of some object-oriented features.

## 2.3. File formats and data structures

### 2.3.1. Overview

In this text detection and recognition test, the input data is a still image.

### 2.3.2. Data structures for encapsulating a single image

An image is provided to the algorithm using the data structure of Table 5.

**Table 5 – Struct representing a single image**

| | C++ code fragment | Remarks |
|---|---|---|
| 1. | `struct Image` | |
| 2. | `{` | |
| 3. | `uint16_t image_width;` | Number of pixels horizontally |
| 4. | `uint16_t image_height;` | Number of pixels vertically |
| 5. | `uint8_t image_depth;` | Number of bits per pixel. Legal values are 8 and 24. |
| 6. | `const uint8_t *data;` | Pointer to raster scanned data. Either RGB color or intensity.<br>If image_depth == 24 this points to 3WH bytes RGBRGBRGB...<br>If image_depth == 8 this points to WH bytes IIIIIII |
| 7. | `};` | |

### 2.3.3. Data structures for reporting detected text

Implementations should report text and its location in each image using the structure of the table below.

**Table 6 – Structure for detected text in a still image**

| | C++ code fragment | Remarks |
|---|---|---|
| 1. | `struct TextOutput` | |
| 2. | `{` | |
| 3. | `bool isAssigned;` | If the text was detected and assigned successfully, this value should be set to true, otherwise false. |
| 4. | `std::string text;` | Characters recognized in a line of connected text |
| 5. | `};` | |

**Table 7 – Structure representing a point in 2D coordinates**

| | C++ code fragment | Remarks |
|---|---|---|
| 1. | `struct Coordinate` | |
| 2. | `{` | |
| 3. | `uint16_t x;` | x-value |
| 4. | `uint16_t y;` | y-value |
| 5. | `};` | |

**Table 8 – Data structure for location information in an image**

| | C++ code fragment | Remarks |
|---|---|---|
| 1. | `using Location = std::vector<Coordinate>;` | In reading order, the coordinates of piecewise line segments drawn through the centroids of the text. When text<br>1. Is just a single character this vector can have size() one indicating a point.<br>2. Appears in a straight line this vector can have size() two, with coordinates giving the end points.<br>Appears in a curve these vectors can have arbitrary length, indicating piecewise lines. |

### 2.3.4. Enumeration of return codes for API function calls

**Table 9 – 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. | `VendorError=5` | Vendor-defined error |
| 7. | `};` | |

# 3. API Specification

## 3.1. Image-to-location

### 3.1.1. Overview

This section defines an API for algorithms that can perform solely text detection. This does not reflect an operational use-case per se, but is included in this evaluation to identify capable algorithms and to support, in-principle, good detection algorithms that have poor recognition capability.

### 3.1.2. API

#### 3.1.2.1. Initialization

Before any text detection calls are made, the NIST test harness will make a call to the initialization of the function in Table 10.

**Table 10 – SDK initialization**

| | | |
|---|---|---|
| Prototype | ReturnCode initialize_text_detector(<br>const std::string &configuration_location) | <br>Input |
| Description | This function initializes the SDK under test. It will be called by the NIST application before any call to detect_text_in_still() is made. | |
| Input Parameters | configuration_location | 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 | |
| ReturnCode Value | Success | Success |
| | ConfigError | Vendor provided configuration files are not readable in the indicated location. |
| | VendorError | Vendor-defined failure |

#### 3.1.2.2. Text detection

The text detection functions of Table 11 accept input imagery and report the location(s) of zero or more lines of text. Text can exist at a point (for a single character), along a straight line, or all a general curve.

**Table 11 – Text detection**

| | | |
|---|---|---|
| Prototypes | ReturnCode detect_text_in_still ( | |
| | const Image &image, | Input |
| | std::vector<Location> &textLocations); | Output |
| Description | This function takes, respectively, a still image and returns the location of lines of text, if any. | |
| Input Parameters | Image | An instance of a Table 5 structure. |
| | | |
| Output Parameters | textLocations | A vector of a Table 8 structure. |
| ReturnCode Value | Success | Success |
| | RefuseInput | Elective refusal to process the input – e.g., because quality is too poor |
| | ExtractError | Involuntary failure to extract features |
| | ParseError | Cannot parse input data (i.e., assertion that input record is non-conformant) |
| | VendorError | Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test. |

## 3.2. Image-to-text with provided location information

### 3.2.1. Overview

This section defines an API for algorithms that perform recognition given text location in an image. This is not a primary operational use-case, but is included for NIST to evaluate the relative difficulties of detection vs. recognition.

### 3.2.2. API

#### 3.2.2.1. Initialization

Before any text recognition calls are made, the NIST test harness will make a call to the initialization of the function in Table 12.

**Table 12 – SDK initialization**

| | | |
|---|---|---|
| Prototype | ReturnCode initialize_text_recognizer( | |
| | const std::string &configuration_location) | Input |
| Description | This function initializes the SDK under test. It will be called by the NIST application before any call to recognize_text_in_still(). | |
| Input Parameters | configuration_location | 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 | |
| ReturnCode Value | Success | Success |
| | ConfigError | Vendor provided configuration files are not readable in the indicated location. |
| | Other | Vendor-defined failure |

#### 3.2.2.2. Text recognition with provided location information

The text recognition functions of Table 13 accept input imagery and locations of text in the image and report zero or more lines of recognized text.

**Table 13 – Text recognition**

| | | | |
|---|---|---|---|
| Prototypes | ReturnCode recognize_text_in_still( | | |
| | const Image &image, | | Input |
| | const std::vector<Location> &textLocation, | | Input |
| | std::vector<TextOutput> &textStrings); | | Output |
| Description | This function takes a still image and K >= 1 locations of text in the image and returns K possibly empty strings of text. The size of textStrings will be pre-allocated to the size of textLocation. textString[k] should be the text associated with textLocation[k]. | | |
| Input Parameters | image | An instance of a Table 5 structure. | |
| | textLocation | A vector of a Table 8 structure. | |
| Output Parameters | textStrings | A vector of a Table 6 structure. | |
| ReturnCode Value | Success | Successful execution | |
| | RefuseInput | Elective refusal to process the input – e.g. because quality is too poor | |
| | ExtractError | Involuntary failure to extract features | |
| | ParseError | Cannot parse input data (i.e. assertion that input record is non-conformant) | |
| | VendorError | Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test. | |

## 3.3. Image-to-text without location information

### 3.3.1. Overview

This section defines an API for algorithms that can perform text recognition in stills. This reflects the primary operational use-case.

### 3.3.2. API

#### 3.3.2.1. Initialization

Before any text recognition/processing calls are made, the NIST test harness will make a call to the initialization of the function in Table 12.

**Table 14 – SDK initialization**

| | | | |
|---|---|---|---|
| Prototype | ReturnCode initialize_text_processor( | | |
| | const std::string &configuration_location) | | Input |
| Description | This function initializes the SDK under test. It will be called by the NIST application before any call to process_text_in_still() is made. | | |
| Input Parameters | configuration_location | 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 | | |
| ReturnCode Value | Success | Success | |
| | ConfigError | Vendor provided configuration files are not readable in the indicated location. | |
| | VendorError | Vendor-defined failure | |

#### 3.3.2.2. Text processing without location information

The text processing functions of Table 15 accept input imagery and report zero or more lines of text.

332 **Table 15 – Text processing**

| | | | |
|---|---|---|---|
| Prototypes | ReturnCode process_text_in_still( | | |
| | const Image &image, | | Input |
| | std::vector<TextOutput> &textStrings); | | Output |
| Description | This function takes a still image and returns strings of text found. | | |
| Input Parameters | image | An instance of a Table 5 structure. | |
| Output Parameters | textStrings | A vector of a Table 6 structure. | |
| ReturnCode Value | Success | Success | |
| | RefuseInput | Elective refusal to process the input – e.g. because quality is too poor | |
| | ExtractError | Involuntary failure to extract features | |
| | ParseError | Cannot parse input data (i.e. assertion that input record is non-conformant) | |
| | VendorError | Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test. | |

333

# Annex A
# Submission of Implementations to the TRAIT 2016

## A.1 Submission of implementations to NIST

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

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

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

## A.2 How to participate

Those wishing to participate in TRAIT 2016 testing must do all of the following, on the schedule listed in this document.

- IMPORTANT: Follow the instructions for cryptographic protection of your SDK and data here. http://www.nist.gov/itl/iad/ig/encrypt.cfm
- Send a signed and fully completed copy of the *Application to Participate in the Text Recognition Algorithm Independent Test (TRAIT) 2016*. This is available at http://www.nist.gov/itl/iad/ig/trait-2016.cfm. This must identify, and include signatures from, the Responsible Parties as defined in the application. The properly signed TRAIT 2016 Application to Participate shall be sent to NIST as a PDF.
- Provide an SDK (Software Development Kit) library which complies with the API (Application Programmer Interface) specified in this document.
    - Encrypted data and SDKs below 20MB can be emailed to NIST at trait2016@nist.gov.
    - Encrypted data and SDKS above 20MB shall be

      EITHER
        - Split into sections AFTER the encryption step. Use the unix "split" commands to make 9MB chunks, and then rename to include the filename extension need for passage through the NIST firewall.
        - `you%  split -a 3 -d -b 9000000  libTRAIT2016_enron_A_02.tgz.gpg`
        - `you%  ls -1 x??? | xargs -iQ  mv Q libTRAIT2016_enron_A_02_Q.tgz.gpg`
        - Email each part in a separate email. Upon receipt NIST will
        - `nist%  cat TRAIT2016_enron_A02_*.tgz.gpg > libTRAIT2016_enron_A_02.tgz.gpg`

      OR
        - Made available as a file.zip.gpg or file.zip.asc download from a generic http webserver[1],

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

| | |
|---|---|
| TRAIT 2016 Test Liaison (A203)<br>100 Bureau Drive<br>A203/Tech225/Stop 8940<br>NIST<br>Gaithersburg, MD 20899-8940<br>USA | In cases where a courier needs a phone number, please use NIST shipping and handling on: 301 -- 975 -- 6296. |

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

## A.3 Implementation validation

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

http://www.nist.gov/itl/iad/ig/trait-2016.cfm shortly after the final evaluation plan is released.

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

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

Software submitted shall implement the TRAIT 2016 API Specification as detailed in the body of this document.

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