

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

# 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](mailto:trait2016@nist.gov)



16

**Provisional Timeline of the TRAIT 2016 Evaluation**

Phase 0 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
	<b>2016-02-08</b>	<b>Last day for submission of algorithms to Phase 1</b>
	2016-03-07	Interim results released to Phase 1 participants
Phase 2	<b>2016-06-12</b>	<b>Last day for submission of algorithms to Phase 2</b>
	2016-07-12	Interim results released to Phase 2 participants
Phase 3	<b>2016-10-19</b>	<b>Last day for submission of algorithms to Phase 3</b>
	2016-Q4	Release of final public report

17

18

19 **Table of Contents**

20 1. TRAIT .....4

21 1.1. Scope .....4

22 1.2. Audience .....4

23 1.3. Market drivers .....4

24 1.4. Test data .....4

25 1.5. Offline testing .....5

26 1.6. Phased testing .....5

27 1.7. Interim reports .....5

28 1.8. Final reports .....5

29 1.9. Application scenarios .....6

30 1.10. Options for participation .....6

31 1.11. Number and schedule of submissions .....6

32 1.12. Core accuracy metrics .....7

33 1.13. Reporting computational efficiency .....7

34 1.14. Hardware specification .....7

35 1.15. Operating system, compilation, and linking environment .....7

36 1.16. Software and Documentation .....8

37 1.17. Runtime behavior .....9

38 1.18. Threaded computations .....9

39 1.19. Time limits .....9

40 2. Data structures supporting the API .....10

41 2.1. Overview .....10

42 2.2. Requirement .....10

43 2.3. File formats and data structures .....10

44 3. API Specification .....11

45 3.1. Image-to-location .....11

46 3.2. Image-to-text with provided location information .....12

47 3.3. Image-to-text without location information .....13

48 Annex A Submission of Implementations to the TRAIT 2016 .....15

49 A.1 Submission of implementations to NIST .....15

50 A.2 How to participate .....15

51 A.3 Implementation validation .....16

52

53 **List of Figures**

54 Figure 1 – Example of inputs and outputs .....4

55

56 **List of Tables**

57 Table 1 – Subtests supported under the TRAIT 2016 activity .....6

58 Table 2 – TRAIT 2016 classes of participation .....6

59 Table 3 – Cumulative total number of algorithms, by class .....6

60 Table 4 – Implementation library filename convention .....8

61 Table 5 – Struct representing a single image .....10

62 Table 6 – Structure for detected text in a still image .....10

63 Table 7 – Structure representing a point in 2D coordinates .....10

64 Table 8 – Data structure for location information in an image .....11

65 Table 9 – Return codes for API function calls .....11

66 Table 10 – SDK initialization .....11

67 Table 11 – Text detection .....12

68 Table 12 – SDK initialization .....12

69 Table 13 – Text recognition .....13

70 Table 14 – SDK initialization .....13

71 Table 15 – Text processing .....14

72 **1. TRAIT**

73 **1.1. Scope**

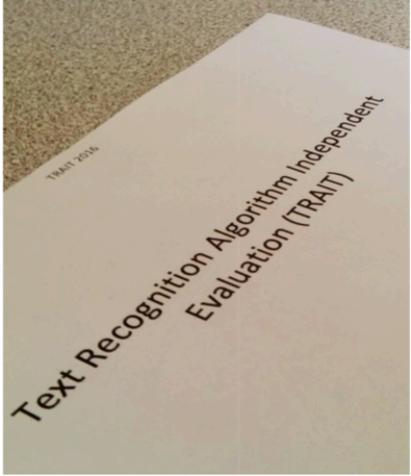
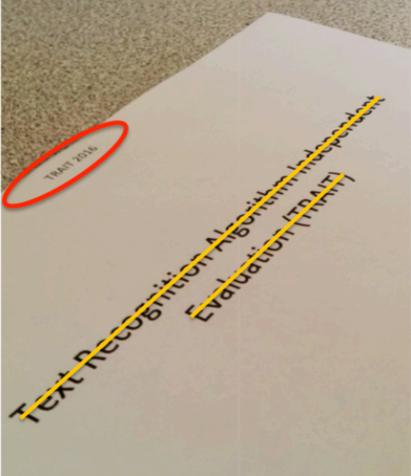
74 This document establishes a concept of operations and C++ API for evaluation of text-in-image detection and recognition  
 75 algorithms submitted to NIST's TRAIT program. See <http://www.nist.gov/itl/iad/ig/trait-2016.cfm> for latest  
 76 documentation.

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

79 **1.2. Audience**

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

84 **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
		<p data-bbox="1040 821 1481 884">First string at very top of page is missed</p> <p data-bbox="1040 947 1481 1094">Text Recognition Algorithm Independent                      x = (200,580)                      y = (160,550)</p> <p data-bbox="1040 1157 1481 1283">Evaluation (TRAIT)                      x = (380,530)                      y = (320, 500)</p>

85  
 86 Organizations will need to implement the API defined in this document. Participation is open worldwide. There is no  
 87 charge for participation. While NIST intends to evaluate technologies that could be readily made operational, the test is  
 88 also open to experimental, prototype and other technologies.

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

92 **1.3. Market drivers**

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

96 **1.4. Test data**

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

98

99 The primary dataset is an operational child exploitation collection containing illicit pornographic images. The images are  
100 present on digital media seized in criminal investigations. The files include children who range in age from infant through  
101 adolescent. Their faces are the subject of a separate face recognition evaluation and development effort (CHEXIA-FACE  
102 2016). Many of the images contain geometrically unconstrained text. This text is human-legible and sometimes has  
103 investigational value. Such text is visible on certificates, posters, logos, uniforms, sports apparel, computer screens,  
104 business cards, newspapers, books lying on tables, cigarette packets and a long list of more rare objects.

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

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

### 111 **1.5. Offline testing**

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

### 116 **1.6. Phased testing**

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

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

### 124 **1.7. Interim reports**

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

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

### 134 **1.8. Final reports**

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

137 Our intention is that the final test reports will publish results for the best-performing implementation from each  
138 participant. Because "best" is ill defined (accuracy vs. processing time, for example), the published reports may include  
139 results for other implementations. The intention is to report results for the most capable implementations (see section  
140 1.12, on metrics). Other results may be included (e.g. in appendices) to show, for example, examples of progress or  
141 tradeoffs. **IMPORTANT: Results will be attributed to the providers.**

142 **1.9. Application scenarios**

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

146 **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.

147

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

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

152 **1.10. Options for participation**

153 The following rules apply:

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

159

**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

160 **1.11. Number and schedule of submissions**

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

163 **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

164 The numbers above may be increased as resources allow.

165 NIST cannot conduct surveys over runtime parameters.

166 **1.12. Core accuracy metrics**

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

172 [1.12a] J. Fiscus, J. Ajot, N. Radde, and C. Laprun, *Multiple Dimension Levenshtein Edit Distance Calculations for Evaluating*  
 173 *Automatic Speech Recognition Systems During Simultaneous Speech*, Proceedings of LREC, 2006.  
 174 [http://www.itl.nist.gov/iad/mig/publications/storage\\_paper/lrec06\\_v0\\_7.pdf](http://www.itl.nist.gov/iad/mig/publications/storage_paper/lrec06_v0_7.pdf)

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

178 [1.12b] C. Wolf and J.-M. Jolion. Object count/Area Graphs for the Evaluation of Object Detection and Segmentation  
 179 Algorithms, International Journal on Document Analysis and Recognition, 8(4):280-296, 2006.  
 180 <http://liris.cnrs.fr/christian.wolf/software/deteval/index.html>

181 **1.13. Reporting computational efficiency**

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

183 **1.14. Hardware specification**

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

186 **1.15. Operating system, compilation, and linking environment**

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

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

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

193 A typical link line might be

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

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

#include <trait2016.h>

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

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

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

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

205 **1.16. Software and Documentation**

206 **1.16.1. SDK Library and Platform Requirements**

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

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

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

216 **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				

217  
 218 NIST will report the size of the supplied libraries.

219 **1.16.2. Configuration and developer-defined data**

220 The implementation under test may be supplied with configuration files and supporting data files. The total size of the  
 221 SDK, that is all libraries, include files, data files and initialization files shall be less than or equal to 1 073 741 824 bytes =  
 222 1024<sup>3</sup> bytes.

223 NIST will report the size of the supplied configuration files.

224 **1.16.3. Installation and Usage**

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

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

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

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

232 **1.16.4. Documentation**

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

236 **1.17. Runtime behavior**

237 **1.17.1. Interactive behavior**

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

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

245 **1.17.2. Exception Handling**

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

248 **1.17.3. External communication**

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

254 **1.17.4. Stateless behavior**

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

260 **1.18. Threaded computations**

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

264 **1.19. Time limits**

265 Given a 12 megapixel input image, the text detection and recognition software should execute in less than 10 seconds.  
266  
267

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

269 **2.1. Overview**

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

272 **2.2. Requirement**

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

275 **2.3. File formats and data structures**

276 **2.3.1. Overview**

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

278 **2.3.2. Data structures for encapsulating a single image**

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

280 **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. If image_depth == 24 this points to 3WH bytes RGBRGBRGB... If image_depth == 8 this points to WH bytes I I I I I I I I
7.	};	

281

282 **2.3.3. Data structures for reporting detected text**

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

284 **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.	};	

285

286 **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.	};	

287

288

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

	C++ code fragment	Remarks
1.	<code>using Location = std::vector&lt;Coordinate&gt;;</code>	<p>In reading order, the coordinates of piecewise line segments drawn through the centroids of the text. When text</p> <ol style="list-style-type: none"> <li>1. Is just a single character this vector can have size() one indicating a point.</li> <li>2. Appears in a straight line this vector can have size() two, with coordinates giving the end points.</li> </ol> <p>Appears in a curve these vectors can have arbitrary length, indicating piecewise lines.</p>

289

290 **2.3.4. Enumeration of return codes for API function calls**

291

**Table 9 – Return codes for API function calls**

	Return code as C++ enumeration	Meaning
	<code>enum class ReturnCode {</code>	
1.	<code>Success=0,</code>	Success
2.	<code>ConfigError=1,</code>	Error reading configuration files
3.	<code>RefuseInput=2,</code>	Elective refusal to process the input
4.	<code>ExtractError=3,</code>	Involuntary failure to process the image
5.	<code>ParseError=4,</code>	Cannot parse the input data
6.	<code>VendorError=5</code>	Vendor-defined error
7.	<code>};</code>	

292

293 **3. API Specification**

294 **3.1. Image-to-location**

295 **3.1.1. Overview**

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

299 **3.1.2. API**

300 **3.1.2.1. Initialization**

301 Before any text detection calls are made, the NIST test harness will make a call to the initialization of the function in Table  
 302 10.

303

**Table 10 – SDK initialization**

Prototype	ReturnCode initialize_text_detector( 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 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

304 **3.1.2.2. Text detection**

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

307 **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.

308 **3.2. Image-to-text with provided location information**

309 **3.2.1. Overview**

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

312 **3.2.2. API**

313 **3.2.2.1. Initialization**

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

316 **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

317 **3.2.2.2. Text recognition with provided location information**

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

320 **Table 13 – Text recognition**

Prototypes	ReturnCode recognize_text_in_still( const Image &image, const std::vector<Location> &textLocation, std::vector<TextOutput> &textStrings);		Input
			Input
			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.	

321 **3.3. Image-to-text without location information**

322 **3.3.1. Overview**

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

325 **3.3.2. API**

326 **3.3.2.1. Initialization**

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

329 **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	

330 **3.3.2.2. Text processing without location information**

331 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, std::vector<TextOutput> &textStrings);	
		Input
		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

334  
335

## Annex A Submission of Implementations to the TRAIT 2016

### 336 A.1 Submission of implementations to NIST

337 NIST requires that all software, data and configuration files submitted by the participants be signed and encrypted.  
 338 Signing is done with the participant's private key, and encryption is done with the NIST public key. The detailed  
 339 commands for signing and encrypting are given here: <http://www.nist.gov/itl/iad/ig/encrypt.cfm>  
 340 NIST will validate all submitted materials using the participant's public key, and the authenticity of that key will be verified  
 341 using the key fingerprint. This fingerprint must be submitted to NIST by writing it on the signed participation agreement.  
 342 By encrypting the submissions, we ensure privacy; by signing the submissions, we ensure authenticity (the software  
 343 actually belongs to the submitter). NIST will reject any submission that is not signed and encrypted. NIST accepts no  
 344 responsibility for anything that is transmitted to NIST that is not signed and encrypted with the NIST public key.

### 345 A.2 How to participate

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

- 347 — IMPORTANT: Follow the instructions for cryptographic protection of your SDK and data here.  
 348 <http://www.nist.gov/itl/iad/ig/encrypt.cfm>
  - 349 — Send a signed and fully completed copy of the *Application to Participate in the Text Recognition Algorithm*  
 350 *Independent Test (TRAIT) 2016*. This is available at <http://www.nist.gov/itl/iad/ig/trait-2016.cfm>. This must identify,  
 351 and include signatures from, the Responsible Parties as defined in the application. The properly signed TRAIT 2016  
 352 Application to Participate shall be sent to NIST as a PDF.
  - 353 — Provide an SDK (Software Development Kit) library which complies with the API (Application Programmer Interface)  
 354 specified in this document.
    - 355 • Encrypted data and SDKs below 20MB can be emailed to NIST at [trait2016@nist.gov](mailto:trait2016@nist.gov).
    - 356 • Encrypted data and SDKS above 20MB shall be
- 357 EITHER
- 358     ▪ Split into sections AFTER the encryption step. Use the unix "split" commands to make 9MB chunks,  
 359     and then rename to include the filename extension need for passage through the NIST firewall.
  - 360     ▪ `you% split -a 3 -d -b 9000000 libTRAIT2016_enron_A_02.tgz.gpg`
  - 361     ▪ `you% ls -l x??? | xargs -iQ mv Q libTRAIT2016_enron_A_02_Q.tgz.gpg`
  - 362     ▪ Email each part in a separate email. Upon receipt NIST will
  - 363     ▪ `nist% cat TRAIT2016_enron_A02_*.tgz.gpg > libTRAIT2016_enron_A_02.tgz.gpg`
- 364 OR
- 365     ▪ Made available as a file.zip.gpg or file.zip.asc download from a generic http webserver<sup>1</sup>,
- 366 OR
- 367     ▪ Mailed as a file.zip.gpg or file.zip.asc on CD / DVD to NIST at this address:

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

<sup>1</sup> NIST will not register, or establish any kind of membership, on the provided website.

**368 A.3 Implementation validation**

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

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

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

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

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

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