

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 0.8, September 8, 2016

Patrick Grother, Mei Ngan, Afzal Godil

Contact via 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
	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	2016-10-31	Last day for submission of algorithms to Phase 3
	2017-Q1	Release of final public report

17

18

19

Updates since the last version are highlighted in **green**.

20

21

Major updates for Phase 3

22

- Removal of algorithm time constraints for text detection and recognition in Section 1.19.

23

- Addition of providing location in the form of a bounding box to algorithms for Class B – Image-to-text with provided location information in Section 3.2.

24

25

- Additional test data characteristics in Section 1.4.

26

- Addition of algorithm support for detection and recognition of URLs in Section 3.3.2.5.

27

- Phase 3 submission deadline is now October 31, 2016.

28

29 **Table of Contents**

30 1. TRAIT5

31 1.1. Scope.....5

32 1.2. Audience5

33 1.3. Market drivers.....6

34 1.4. Test data7

35 1.5. Offline testing7

36 1.6. Phased testing.....7

37 1.7. Interim reports.....7

38 1.8. Final reports.....8

39 1.9. Application scenarios8

40 1.10. Options for participation.....8

41 1.11. Number and schedule of submissions8

42 1.12. Core accuracy metrics9

43 1.13. Reporting computational efficiency.....9

44 1.14. Hardware specification9

45 1.15. Operating system, compilation, and linking environment.....9

46 1.16. Software and Documentation10

47 1.17. Runtime behavior.....11

48 1.18. Threaded computations.....12

49 1.19. Time limits.....12

50 2. Data structures supporting the API.....13

51 2.1. Namespace.....13

52 2.2. Overview13

53 2.3. Requirement13

54 2.4. File formats and data structures.....13

55 3. API Specification.....15

56 3.1. Image-to-location.....15

57 3.2. Image-to-text with provided location information17

58 3.3. Image-to-text-and-location.....19

59 Annex A Submission of Implementations to the TRAIT 201622

60 A.1 Submission of implementations to NIST.....22

61 A.2 How to participate22

62 A.3 Implementation validation23

64 **List of Figures**

65 Figure 1 – Example of inputs and outputs5

66 Figure 2 – Example of input with provided location information and outputs.....6

67

68 **List of Tables**

69 Table 1 – Subtests supported under the TRAIT 2016 activity8

70 Table 2 – TRAIT 2016 classes of participation.....8

71 Table 3 – Cumulative total number of algorithms, by class.....9

72 Table 4 – Implementation library filename convention10

73 Table 5 – Struct representing a single image.....13

74 Table 7 – Location Types.....13

75 Table 8 – Data structure for location information in an image.....14

76 Table 9 – Enumeration of return codes for API function calls.....14

77 Table 10 – ReturnStatus structure.....14

78 Table 11 – SDK initialization.....16

79 Table 12 – GPU index specification.....16

80 Table 13 – Text detection16

81 Table 14 – SDK initialization.....18

82 Table 15 – GPU index specification18
83 Table 16 – Text recognition18
84 Table 17 – SDK initialization.....20
85 Table 18 – GPU index specification20
86 Table 19 – Text processing.....20
87 Table 20 – URL processing21
88
89

90 **1. TRAIT**

91 **1.1. Scope**

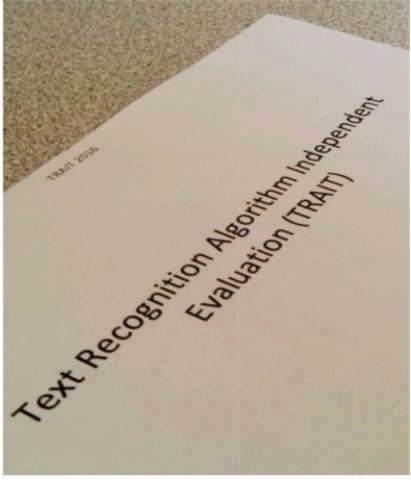
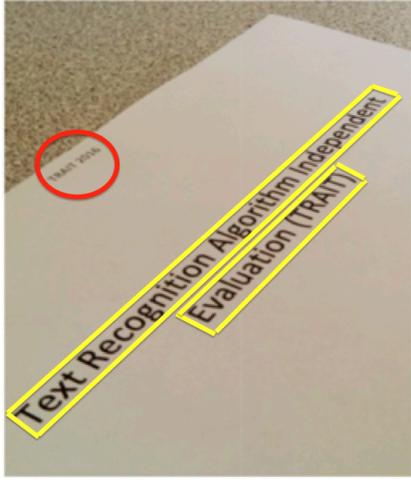
92 This document establishes a concept of operations and C++ API for evaluation of text-in-image detection and recognition
 93 algorithms submitted to NIST's TRAIT program. See <http://www.nist.gov/itl/iad/ig/trait-2016.cfm> for latest
 94 documentation.

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

97 **1.2. Audience**

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

102
 103 **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>First string at very top of page is missed</p> <p>Text Recognition Algorithm Independent $(x,y) = (10,10),(10,100),(20,100),(20,10),(10,10)$</p> <p>Evaluation (TRAIT) $(x,y) = (30,30),(30,60),(40,60),(30,40),(30,30)$</p>

104
 105
 106
 107
 108
 109
 110
 111
 112
 113

114

115

Figure 2 – Example of input with provided location information and outputs

<p>A simple example of out-of-plane text with provided location information marked by the yellow lines.</p> <p>NOTE: When location information is provided to a text recognition algorithm, we provide a simple line instead of a polygon. The reason for this is that our ground truthing process put emphasis on maximizing the number of images i.e. high speed, high volume. This meant that drawing bounding box polygons, which is slow, was deprecated in favor of drawing lines. Implementations that require polygons for recognition will have to estimate that information starting from the provided line.</p>	<p>Possible Output text corresponding to provided locations</p>
	<p>IN CASE OF FIRE DO NOT USE ELEVATOR USE EXIT STAIRS</p>

116

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

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

123 **1.3. Market drivers**

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

127 **1.4. Test data**

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

129
 130 The primary dataset is an operational child exploitation collection containing illicit pornographic images. The images are
 131 present on digital media seized in criminal investigations. The files include children who range in age from infant through
 132 adolescent. Their faces are the subject of a separate face recognition evaluation and development effort (CHEXIA-FACE
 133 2016). Many of the images contain geometrically unconstrained text. This text is human-legible and sometimes has
 134 investigational value. Such text is visible on certificates, posters, logos, uniforms, sports apparel, computer screens,
 135 business cards, newspapers, books lying on tables, cigarette packets and a long list of more rare objects. **There will also**
 136 **be instances where watermarks or logos were post-processed into the image.**

137 The text is most commonly in English with French, Spanish, German and Cyrillic present in significant quantity. We do not
 138 intend to test non-Roman alphabets. **Images contain text on clothing, low-resolution text, exotic fonts, symbols, and**
 139 **numbers.**

140 **URLs: A quarter of the test images contain URLs, often in-plane, post-processed into the image. URLs can provide**
 141 **important leads to an investigation. As such, a new function to specifically evaluate processing of URLs has been added to**
 142 **this API document. See Sections 3.3.2.5 and 1.12 for details.**

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

147 **1.5. Offline testing**

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

153 **1.6. Phased testing**

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

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

161 **1.7. Interim reports**

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

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

171 **1.8. Final reports**

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

174 Our intention is that the final test reports will publish results for the best-performing implementation from each
175 participant. Because “best” is ill defined (accuracy vs. processing time, for example), the published reports may include
176 results for other implementations. The intention is to report results for the most capable implementations (see section
177 1.12, on metrics). Other results may be included (e.g. in appendices) to show, for example, examples of progress or
178 tradeoffs. **IMPORTANT: Results will be attributed to the providers.**

179 **1.9. Application scenarios**

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

183 **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-and-location
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(s)	Image(s) and location(s) of text	Image(s)
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 along with their corresponding locations in the image.

184

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

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

189 **1.10. Options for participation**

190 The following rules apply:

- 191 – A participant must properly follow, complete and submit the TRAIT 2016 Participation Application (see Annex A). This
192 must be done once, not before December 1, 2015. It is not necessary to do this for each submitted SDK.
- 193 – Participants may submit class C algorithms only if at least 1 class B algorithm is also submitted.
- 194 – All submissions shall implement exactly one of the functionalities defined in Table 2. A library shall not implement
195 two or more classes.

196

Table 2 – TRAIT 2016 classes of participation

Function	Image-to-location	Image-to-text with provided location information	Image-to-text-and-location
Class label	A	B	C
Must also submit to class			B
API requirements	3.1	3.2	3.3

197 **1.11. Number and schedule of submissions**

198 The test is conducted in three phases, separated by a few months. The maximum total (i.e. cumulative) number of
199 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-and-location	2	4	6

201 The numbers above may be increased as resources allow.

202 NIST cannot conduct surveys over runtime parameters.

203 **1.12. Core accuracy metrics**

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

209 [1.12a] J. Fiscus, J. Ajot, N. Radde, and C. Laprun, *Multiple Dimension Levenshtein Edit Distance Calculations for Evaluating*
 210 *Automatic Speech Recognition Systems During Simultaneous Speech*, Proceedings of LREC, 2006.
 211 http://www.itl.nist.gov/iad/mig/publications/storage_paper/lrec06_v0_7.pdf

212 **Detection:** The text detection task will be evaluated, somewhat similar to prior open evaluations [1.12b]. However, in our
 213 case the ground truth text is defined by both bounding boxes and line segments. Hence our methodology will be based on
 214 the intersection of ground truth and detection bounding boxes and the matching distance between the ground truth lines
 215 and the detection-bounding boxes as the criteria.

216 [1.12b] C. Wolf and J.-M. Jolion. Object count/Area Graphs for the Evaluation of Object Detection and Segmentation
 217 Algorithms, International Journal on Document Analysis and Recognition, 8(4):280-296, 2006.
 218 <http://liris.cnrs.fr/christian.wolf/software/deteval/index.html>

219 **URLs:** Operationally, the most pertinent part of a URL is the domain name. In evaluating URL text recognition, we may
 220 consider the accuracy impact of ignoring punctuation, (e.g., { . , : / ? @ , etc.}), protocol text, (e.g., http, https, etc.), and
 221 top level domain words (e.g., com, org, info, net, etc.).

222 **1.13. Reporting computational efficiency**

223 NIST will also report timing statistics for all core functions of the submitted SDK implementations. **All timing tests will be**
 224 **executed on unloaded machines running a single process.**

225 **1.14. Hardware specification**

226 NIST intends to execute the software on Dual Intel Xeon E5-2695 3.3 GHz CPUs (14 cores each; 56 logical CPUs total) with
 227 Dual NVIDIA Tesla K40 **Graphics Processing Units** (GPUs). NIST will respond to prospective participants' questions on the
 228 hardware by amending this section.

229 **1.15. Operating system, compilation, and linking environment**

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

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

234 NIST will link the provided library file(s) to our C++ language test drivers. Participants are required to provide their library
 235 in a format that is linkable using the C++11 compiler, g++ version 4.8.3.

236 A typical link line might be

237 `g++ -std=c++11 -l. -Wall -m64 -o trait16test trait16test.cpp -L. -ltrait2016_Enron_A_07`

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

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

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

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

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

246 Dependencies on external dynamic/shared libraries such as compiler-specific development environment libraries are
 247 discouraged. If absolutely necessary, external libraries must be provided to NIST upon prior approval by the Test Liaison.
 248 Libraries to access the GPU must be provided to NIST as a part of the algorithm submission package.

249 **1.16. Software and Documentation**

250 **1.16.1. SDK Library and Platform Requirements**

251 Participants shall provide NIST with binary code only (i.e., no source code). Header files (".h") are allowed, but these shall
 252 not contain intellectual property of the company nor any material that is otherwise proprietary. The SDK should be
 253 submitted in the form of a dynamically linked library file. A separate library file shall be submitted for each class of
 254 participation (i.e., CLASS_A, CLASS_B, CLASS_C).

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

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

261 **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				

262
 263 NIST will report the size of the supplied libraries.

264 **1.16.2. Configuration and developer-defined data**

265 The implementation under test may be supplied with configuration files and supporting data files. The total size of the
 266 SDK, that is all libraries, include files, data files and initialization files shall be less than or equal to 1 073 741 824 bytes =
 267 1024³ bytes.

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

269 **1.16.3. Submission folder hierarchy**

270 Participant submissions should contain the following folders at the top level

- 271 • lib/ - contains all participant-supplied software libraries
- 272 • config/ - contains all configuration and developer-defined data
- 273 • doc/ - contains any participant-provided documentation regarding the submission

274 **1.16.4. Installation and Usage**

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

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

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

281 **1.16.5. Documentation**

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

285 **1.17. Runtime behavior**

286 **1.17.1. Interactive behavior**

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

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

294 **1.17.2. Exception Handling**

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

297 **1.17.3. External communication**

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

303 **1.17.4. Stateless behavior**

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

309 **1.18. Threaded computations**

310 All implementations should run without threads, or with exactly one single process. This allows NIST to parallelize the test
311 by dividing the workload across multiple cores and multiple machines. **The implementation shall be tolerant of NIST**
312 **running $N > 2$ processes concurrently.** NIST's calling applications are single-threaded.

313 **1.19. Time limits**

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

315

316

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

318 **2.1. Namespace**

319 All data structures and API interfaces/function calls will be declared in the TRAIT2016 namespace.

320 **2.2. Overview**

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

323 **2.3. Requirement**

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

326 **2.4. File formats and data structures**

327 **2.4.1. Overview**

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

329 **2.4.2. Data structures for encapsulating a single image**

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

331 **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.	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.	};	

332

333 **2.4.3. Data structures for reporting detected text**

334 Implementations should report locations of text in each image using the structure of the table below.

335

336 **Table 6 – 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.	};	

337

338 **Table 7 – Location Types**

	Location Type as C++ enumeration	Meaning
	enum class LocationType {	

1.	<code>/* Input only */ Line=1,</code>	Coordinates of line segments drawn through the centroids of the text. There will be N=2 points with coordinates giving the endpoints. This type is used for input to the implementation only, i.e. Class B (Image-to-text with provided location).
2.	<code>/* Input and Output */ Polygon=2</code>	
3.	<code>};</code>	In reading, clockwise order, the coordinates of a closed polygon drawn around the line of text, where the coordinates of the first and last points in the polygon are the same. This type is used for input, i.e. Class B (Image-to-text with provided location) and output from the implementation, i.e. Class A (Image-to-location) and Class C (Image-to-text-and-location).

339

340

Table 8 – Data structure for location information in an image

	C++ code fragment	Remarks
1.	<code>struct Location</code>	
2.	<code>{</code>	
3.	<code>std::vector<Coordinate> points;</code>	Coordinates representing the location of the text
4.	<code>LocationType type;</code>	For Class B (Image-to-text with provided location), this will provide the location information for text in the form of a line through the centroids of the text or a polygon where the text is known to be inside. For Class A (Image-to-location) and C (Image-to-text-and-location), the locations returned by the implementation MUST be a Polygon.
5.	<code>};</code>	

341

2.4.4. Data structure for return value of API function calls

342

Table 9 – Enumeration of 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>VendorError=3,</code>	Vendor-defined error
5.	<code>NotImplemented=4</code>	Function is not implemented by provider
6.	<code>};</code>	

343

Table 10 – ReturnStatus structure

	C++ code fragment	Meaning
	<code>struct ReturnStatus {</code>	
1.	<code>TRAIT2016::ReturnCode code;</code>	Return Code
2.	<code>std::string info;</code>	Optional information string
3.	<code>// constructors</code>	
4.	<code>};</code>	

344 **3. API Specification**

345 **3.1. Image-to-location**

346 **3.1.1. Overview**

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

350 **3.1.2. API**

351 **3.1.2.1. Interface**

352 The Class A software under test must implement the interface `ImgToLocationInterface` by subclassing this class
 353 and implementing each method specified therein.

	C++ code fragment	Remarks
1.	<code>class ImgToLocationInterface</code>	
2.	<code>{</code> <code>public:</code>	
3.	<code>virtual ReturnStatus initialize_text_detector(const std::string &configuration_location);</code>	
4.	<code>virtual ReturnStatus detect_text_in_still(const Image &image, std::vector<Location> &textLocations) = 0;</code>	
5.	<code>virtual void set_gpu(uint8_t gpuNum) = 0;</code>	
6.	<code>static ClassAImplPtr getImplementation();</code>	Factory method to return a managed pointer to the <code>ImgToLocationInterface</code> object. This function is implemented by the submitted library and must return a managed pointer to the <code>ImgToLocationInterface</code> object.
7.	<code>};</code>	

354
 355 There is one class (static) method declared in `ImgToLocationInterface`, `getImplementation()` which must
 356 also be implemented by the SDK. This method returns a shared pointer to the object of the interface type, an instantiation
 357 of the implementation class. A typical implementation of this method is also shown below as an example.
 358

	C++ code fragment	Remarks
	<code>#include "trait2016NullImplClassA.h"</code>	
	<code>using namespace TRAIT2016;</code>	
	<code>NullImplClassA::NullImplClassA() { }</code>	
	<code>NullImplClassA::~NullImplClassA() { }</code>	
	<code>ClassAImplPtr ImgToLocationInterface::getImplementation() { NullImplClassA *p = new NullImplClassA(); ClassAImplPtr ip(p); return (ip); }</code>	
	<code>// Other implemented functions</code>	

359 **3.1.2.2. Initialization**

360 Before any text detection calls are made, the NIST test harness will make a call to the initialization of the function in Table
 361 11.

362 **Table 11 – SDK initialization**

Prototype	ReturnStatus 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.
Return Code	Success	Success
	ConfigError	Vendor provided configuration files are not readable in the indicated location.
	VendorError	Vendor-defined failure

363 **3.1.2.3. GPU Index Specification**

364 For implementations using GPUs, the function of Table 12 specifies a sequential index for which GPU device to execute
 365 on. This enables the test software to orchestrate load balancing across multiple GPUs.

366 **Table 12 – GPU index specification**

Prototypes	void set_gpu (uint8_t gpuNum);	Input
Description	This function sets the GPU device number to be used by all subsequent implementation function calls. gpuNum is a zero-based sequence value of which GPU device to use. 0 would mean the first detected GPU, 1 would be the second GPU, etc. If the implementation does not use GPUs, then this function call should simply do nothing.	
Input Parameters	gpuNum	Index number representing which GPU to use.

367 **3.1.2.4. Text detection**

368 The text detection functions of Table 13 accept input imagery and report the location(s) of zero or more lines of text, in
 369 the form of a closed polygon. NIST may evaluate on images with no text in them.

370 **Table 13 – Text detection**

Prototypes	ReturnStatus detect_text_in_still (const Image &image, std::vector<Location> &textLocations);	Input Output
Description	This function takes, respectively, a still image and returns the locations of lines of text, if any, in the form of closed polygon(s). The output textLocations vector will initially be empty. It is up to the implementation to populate the vector and ensure textStrings.size() == images.size.	
Input Parameters	Image	A Table 5 structure
Output Parameters	textLocations	A vector of Table 8 structure
Return Code	Success	Success
	RefuseInput	Elective refusal to process the input – e.g., because quality is too poor
	VendorError	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

371 **3.2. Image-to-text with provided location information**

372 **3.2.1. Overview**

373 This section defines an API for algorithms that perform recognition given text location in an image. This is not a primary
 374 operational use-case, but is included for NIST to evaluate the relative difficulties of detection vs. recognition. The
 375 location(s) of the text will be provided to the algorithm either in the form of a line through the centroids of the text or a
 376 closed bounding box polygon around the text. Most of the text locations provided via bounding boxes contain a single
 377 line of text, but some may contain multiple lines of text. The bounding boxes aren't necessarily tight around the text.

378 **3.2.2. API**

379 **3.2.2.1. Interface**

380 The Class B software under test must implement the interface `ImgToTextInterface` by subclassing this class and
 381 implementing each method specified therein. See

C++ code fragment	Remarks
1. <code>class ImgToTextInterface</code>	
2. <code>{</code> <code>public:</code>	
3. <code>virtual ReturnStatus initialize_text_recognizer(const std::string &configuration_location);</code>	
4. <code>virtual ReturnStatus recognize_text_in_still(const Image &image, const std::vector<Location> &textLocations, std::vector<std::string> &textStrings) = 0;</code>	
5. <code>virtual void set_gpu(uint8_t gpuNum) = 0;</code>	
6. <code>static ClassBImplPtr getImplementation();</code>	Factory method to return a managed pointer to the <code>ImgToTextInterface</code> object. This function is implemented by the submitted library and must return a managed pointer to the <code>ImgToTextInterface</code> object.
7. <code>};</code>	

382 There is one class (static) method declared in `ImgToTextInterface`, `getImplementation()` which must also be
 383 implemented by the SDK. This method returns a shared pointer to the object of the interface type, an instantiation of the
 384 implementation class. A typical implementation of this method is also shown below as an example.
 385
 386

C++ code fragment	Remarks
<pre> #include "trait2016NullImplClassB.h" using namespace TRAIT2016; NullImplClassB::NullImplClassB() { } NullImplClassB::~NullImplClassB() { } ClassBImplPtr ImgToTextInterface::getImplementation() { NullImplClassB *p = new NullImplClassB(); ClassBImplPtr ip(p); return (ip); } // Other implemented functions </pre>	

387

388 **3.2.2.2. Initialization**

389 Before any text recognition calls are made, the NIST test harness will make a call to the initialization of the function in
 390 Table 14.

391 **Table 14 – SDK initialization**

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

392 **3.2.2.3. GPU Index Specification**

393 For implementations using GPUs, the function of Table 15 specifies a sequential index for which GPU device to execute
 394 on. This enables the test software to orchestrate load balancing across multiple GPUs.

395 **Table 15 – GPU index specification**

Prototypes	<code>void set_gpu (uint8_t gpuNum);</code>	Input
Description	This function sets the GPU device number to be used by all subsequent implementation function calls. <code>gpuNum</code> is a zero-based sequence value of which GPU device to use. 0 would mean the first detected GPU, 1 would be the second GPU, etc. If the implementation does not use GPUs, then this function call should simply do nothing.	
Input Parameters	<code>gpuNum</code>	Index number representing which GPU to use.

396 **3.2.2.4. Text recognition with provided location information**

397 The text recognition functions of Table 16 accept input imagery and locations of text in the image and report zero or more
 398 lines of recognized text. NIST may provide locations where no text exists, and the implementation should handle that.

399 **Table 16 – Text recognition**

Prototypes	<code>ReturnStatus recognize_text_in_still(const Image &image, const std::vector<Location> &textLocations, std::vector<std::string> &textStrings);</code>	Input Input Output
Description	This function takes a still images and $K \geq 1$ locations of text and returns K (possibly empty) strings of text for the image. The output vector will initially be empty and it is up to the implementation to ensure that <code>textStrings.size() == images.size()</code> . <code>textStrings[i]</code> should correspond to the location information from <code>textLocations[i]</code> . If no text is recognized for a particular provided location, the implementation shall populate <code>textStrings[i]</code> with an empty string (i.e., ""). The provided location information will be in the form of a line, that is, <code>textLocations[i].type=Line</code> .	
Input Parameters	<code>image</code>	A Table 5 structure
	<code>textLocations</code>	A vector of Table 8 structure
Output Parameters	<code>textStrings</code>	A vector of <code>std::string</code>
Return Code	Success	Success
	RefuseInput	Elective refusal to process the input – e.g. because quality is too poor
	VendorError	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

400 **3.3. Image-to-text-and-location**

401 **3.3.1. Overview**

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

404 **3.3.2. API**

405 **3.3.2.1. Interface**

406 The Class C software under test must implement the interface `ImgToTextAndLocInterface` by subclassing this class
407 and implementing each method specified therein. See

C++ code fragment	Remarks
1. <code>class ImgToTextAndLocInterface</code>	
2. <code>{</code>	
3. <code>public:</code>	
3. <code>virtual ReturnStatus initialize_text_processor(const std::string &configuration_location, bool &useGPU);</code>	
4. <code>virtual ReturnStatus process_text_in_still(const Image &image, std::vector<Location> &textLocations, std::vector<std::string> &textStrings) = 0;</code>	
5. <code>virtual void set_gpu(uint8_t gpuNum) = 0;</code>	
6. <code>virtual ReturnStatus process_urls_in_still(const Image &image, double &confidenceScore, std::vector<Location> &urlLocations, std::vector<std::string> &urlStrings);</code>	
7. <code>static ClassCImplPtr getImplementation();</code>	Factory method to return a managed pointer to the <code>ImgToTextAndLocInterface</code> object. This function is implemented by the submitted library and must return a managed pointer to the <code>ImgToTextAndLocInterface</code> object.
8. <code>};</code>	

408
409 There is one class (static) method declared in `ImgToTextAndLocInterface`, `getImplementation()` which
410 must also be implemented by the SDK. This method returns a shared pointer to the object of the interface type, an
411 instantiation of the implementation class. A typical implementation of this method is also shown below as an example.
412

C++ code fragment	Remarks
-------------------	---------

```
#include "trait2016NullImplClassC.h"

using namespace TRAIT2016;

NullImplClassC::NullImplClassC() { }

NullImplClassC::~NullImplClassC() { }

ClassCImplPtr
ImgToTextAndLocInterface::getImplementation()
{
    NullImplClassC *p = new NullImplClassC();
    ClassCImplPtr ip(p);
    return (ip);
}

// Other implemented functions
```

413

414 **3.3.2.2. Initialization**

415 Before any text recognition/processing calls are made, the NIST test harness will make a call to the initialization of the
416 function in Table 14.

417 **Table 17 – SDK initialization**

Prototype	ReturnStatus 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.
Return Code	Success	Success
	ConfigError	Vendor provided configuration files are not readable in the indicated location.
	VendorError	Vendor-defined failure

418 **3.3.2.3. GPU Index Specification**

419 For implementations using GPUs, the function of Table 18 specifies a sequential index for which GPU device to execute
420 on. This enables the test software to orchestrate load balancing across multiple GPUs.

421 **Table 18 – GPU index specification**

Prototypes	void set_gpu (uint8_t gpuNum);	Input
Description	This function sets the GPU device number to be used by all subsequent implementation function calls. gpuNum is a zero-based sequence value of which GPU device to use. 0 would mean the first detected GPU, 1 would be the second GPU, etc. If the implementation does not use GPUs, then this function call should simply do nothing.	
Input Parameters	gpuNum	Index number representing which GPU to use.

422 **3.3.2.4. Text processing without location information**

423 The text processing function of Table 19 accepts input imagery and report zero or more lines of text.

424 **Table 19 – Text processing**

Prototypes	ReturnStatus process_text_in_still(const Image &image, std::vector<Location> &textLocations, std::vector<std::string> &textStrings);	Input
		Output
		Output

Description	This function takes a still image and returns strings of text and their corresponding locations found in the image. <code>textStrings[i]</code> shall correspond to the location(s) for text in <code>textLocations[i]</code> . The output vectors will initially be empty and it is up to the implementation to ensure that <code>textLocations.size()==textStrings.size()</code> . The implementation shall report the location of lines of text in the form of a closed polygon, that is, <code>textLocations[i].type=Polygon</code> .	
Input Parameters	image	A Table 5 structure
Output Parameters	textLocations	A vector of Table 8 structure
	textStrings	A vector of <code>std::string</code>
Return Code	Success	Success
	RefuseInput	Elective refusal to process the input – e.g. because quality is too poor
	VendorError	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

425 **3.3.2.5. URL detection and recognition**

426 The URL processing function of Table 20 accepts input imagery and report zero or more lines of URL text along with a URL
 427 detection confidence score. Implementation of this function is optional but encouraged.

428 **Table 20 – URL processing**

Prototypes	<code>ReturnStatus process_urls_in_still(</code> <code>const Image &image,</code> <code>double &confidenceScore,</code> <code>std::vector<Location> &urlLocations,</code> <code>std::vector<std::string> &urlStrings);</code>	Input Output Output Output
Description	This function takes a still image and returns zero or more strings of URL text and their corresponding locations found in the image, along with a URL detection confidence score on [0,1], with 0 indicating certainty that the image contains no URL text, and 1 indicating certainty that image contains URL text. The score will be used to perform threshold-based analysis of algorithm URL detection performance. <code>urlStrings[i]</code> shall correspond to the location(s) for URL text in <code>urlLocations[i]</code> . The output vectors will initially be empty and it is up to the implementation to ensure that <code>textLocations.size()==textStrings.size()</code> . The implementation shall report the location of lines of text in the form of a closed polygon, that is, <code>textLocations[i].type=Polygon</code> .	
Input Parameters	image	A Table 5 structure
Output Parameters	confidenceScore	A measure of certainty that the image contains a URL. This value shall be on [0,1], with 0 indicating certainty of no URL text in image and 1 indicating certainty of the existence of URL text in image
	urlLocations	A vector of Table 8 structure
	urlStrings	A vector of <code>std::string</code>
Return Code	Success	Success
	RefuseInput	Elective refusal to process the input – e.g. because quality is too poor
	VendorError	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

429

430
431

Annex A Submission of Implementations to the TRAIT 2016

432 A.1 Submission of implementations to NIST

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

441 A.2 How to participate

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

- 443 — IMPORTANT: Follow the instructions for cryptographic protection of your SDK and data here.
 444 <http://www.nist.gov/itl/iad/ig/encrypt.cfm>
- 445 — Send a signed and fully completed copy of the *Application to Participate in the Text Recognition Algorithm*
 446 *Independent Test (TRAIT) 2016*. This is available at <http://www.nist.gov/itl/iad/ig/trait-2016.cfm>. This must identify,
 447 and include signatures from, the Responsible Parties as defined in the application. The properly signed TRAIT 2016
 448 Application to Participate shall be sent to NIST as a PDF.
- 449 — Provide an SDK (Software Development Kit) library which complies with the API (Application Programmer Interface)
 450 specified in this document.
 - 451 • Encrypted data and SDKs below 20MB can be emailed to NIST at trait2016@nist.gov.
 - 452 • Encrypted data and SDKS above 20MB shall be
 - 453 EITHER
 - 454 ▪ Split into sections AFTER the encryption step. Use the unix "split" commands to make 9MB chunks,
 455 and then rename to include the filename extension need for passage through the NIST firewall.
 - 456 ▪ `you% split -a 3 -d -b 9000000 libTRAIT2016_enron_A_02.tgz.gpg`
 - 457 ▪ `you% ls -l x??? | xargs -iQ mv Q libTRAIT2016_enron_A_02_Q.tgz.gpg`
 - 458 ▪ Email each part in a separate email. Upon receipt NIST will
 - 459 ▪ `nist% cat TRAIT2016_enron_A02_*.tgz.gpg > libTRAIT2016_enron_A_02.tgz.gpg`
 - 460 OR
 - 461 ▪ Made available as a file.zip.gpg or file.zip.asc download from a generic http webserver¹,
 - 462 OR
 - 463 ▪ 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.
--	---

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

464 A.3 Implementation validation

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

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

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

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

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

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