## 4 Software and Documentation

4.1 Library and Platform Requirements ................................. 20
4.2 Configuration and Vendor-Defined Data .............................. 20
4.3 Linking ........................................................................ 20
4.4 Installation and Usage .................................................. 21
4.5 Modes of Operation ...................................................... 21
4.6 Runtime Behavior .......................................................... 21
4.6.1 Interactive Behavior .................................................. 21
4.6.2 Status Messages ....................................................... 21
4.6.3 Error Codes and Exception Handling ......................... 21
4.6.4 External Communication .......................................... 22
4.6.5 Stateful Behavior ...................................................... 22

## 5 How to Submit Implementations to FpVTE2012

5.1 Confidentiality and Integrity Protection ............................. 23
5.2 How to Participate ........................................................ 23
5.3 Implementation Validation ............................................. 24

## A Namespace Index

A.1 Namespace List ............................................................ 26

## B Class Index

B.1 Class Hierarchy ........................................................... 27

## C Class Index

C.1 Class List ...................................................................... 28

## D Namespace Documentation

D.1 BiometricEvaluation::Error Namespace Reference ............. 30
   D.1.1 Detailed Description .............................................. 31
   D.1.2 Function Documentation ........................................ 31
   D.1.2.1 errorStr ......................................................... 31
D.2 BiometricEvaluation::Finger Namespace Reference ........... 31
   D.2.1 Detailed Description .............................................. 32
D.3 BiometricEvaluation::Image Namespace Reference .......... 32
   D.3.1 Detailed Description .............................................. 33
   D.3.2 Function Documentation ........................................ 33
   D.3.2.1 operator<< ...................................................... 33
   D.3.2.2 distance ......................................................... 33
D.4 BiometricEvaluation::IO Namespace Reference ............... 33
   D.4.1 Detailed Description .............................................. 34
   D.4.2 Typedef Documentation ......................................... 34
   D.4.2.1 PropertiesMap ................................................ 34
D.5 BiometricEvaluation::IO::Utility Namespace Reference .... 34
   D.5.1 Detailed Description .............................................. 35
   D.5.2 Function Documentation ........................................ 35
   D.5.2.1 removeDirectory ............................................. 35
   D.5.2.2 setAsideName .................................................. 35
   D.5.2.3 getFileSize ..................................................... 36
   D.5.2.4 fileExists ....................................................... 36

3.5.3.3 Error::Exception .................................................. 19

D.5.2.4 fileExists ....................................................... 36
D.5.2.5 validateRootName ............................................. 36
D.5.2.6 constructAndCheckPath .................................. 37
D.5.2.7 makePath ................................................. 37
D.5.2.8 readFile .................................................... 37
D.5.2.9 writeFile ................................................... 38
D.5.2.10 writeFile ................................................... 38
D.5.2.11 isReadable ............................................... 38
D.5.2.12 isWritable ............................................... 39
D.5.2.13 createTemporaryFile ................................... 39
D.5.2.14 createTemporaryFile ................................... 39
D.6 BiometricEvaluation::Text Namespace Reference .......... 40
D.6.1 Detailed Description ........................................ 40
D.6.2 Function Documentation .................................... 40
D.6.2.1 dirname .................................................... 40
D.7 FPVTE2012 Namespace Reference ............................. 40
D.7.1 Detailed Description ........................................ 41
D.7.2 Typedef Documentation ..................................... 41
D.7.2.1 FingerImageSet ......................................... 41
D.7.2.2 FingerOutputSet ......................................... 41
D.7.2.3 CandidateSet .............................................. 41
D.7.3 Function Documentation .................................... 42
D.7.3.1 operator<< ............................................... 42
D.7.3.2 operator<< ............................................... 42
D.7.3.3 operator<< ............................................... 42
E Class Documentation ............................................. 43
E.1 BiometricEvaluation::Memory::AutoArray<T> Class Template Reference ................................. 43
E.1.1 Detailed Description .................................... 44
E.1.2 Member Typedef Documentation ............................. 44
E.1.2.1 value_type .............................................. 44
E.1.2.2 size_type ................................................. 44
E.1.2.3 iterator .................................................. 44
E.1.2.4 const_iterator ......................................... 44
E.1.2.5 reference ............................................... 44
E.1.2.6 const_reference ........................................ 45
E.1.3 Constructor & Destructor Documentation ................ 45
E.1.3.1 AutoArray ............................................... 45
E.1.3.2 AutoArray ............................................... 45
E.1.3.3 ~AutoArray ............................................. 45
E.1.4 Member Function Documentation ........................ 45
E.1.4.1 operator T * ........................................... 45
E.1.4.2 operator const T * ..................................... 46
E.1.4.3 operator[] ............................................... 46
E.1.4.4 operator[] ............................................... 46
E.1.4.5 at ......................................................... 46
E.1.4.6 at ......................................................... 47
E.1.4.7 begin ..................................................... 47
E.1.4.8 begin ..................................................... 47
E.1.4.9 end ......................................................... 47
E.1.4.10 end ....................................................... 47
E.1.4.11 size ....................................................... 48
E.1.4.12 resize .................................................... 48
E.1.4.13 copy ..................................................... 48
E.1.4.14 copy ..................................................... 49
E.1.4.15 operator= .............................................. 49
E.30.1.2 initEnrollmentTemplateExtraction ......................................................... 100
E.30.1.3 makeEnrollmentTemplate ................................................................. 101
E.30.1.4 finalizeEnrollment .............................................................................. 101
E.30.1.5 initSearchTemplateExtraction ............................................................. 102
E.30.1.6 makeSearchTemplate ........................................................................... 102
E.30.1.7 initIdentificationStageOne ................................................................. 103
E.30.1.8 identifyTemplateStageOne ................................................................. 103
E.30.1.9 initIdentificationStageTwo ................................................................. 104
E.30.1.10 identifyTemplateStageTwo ............................................................... 104
E.30.1.11 getSDK ............................................................................................... 105

E.31 BiometricEvaluation::Image::Size Struct Reference ..................................... 105
  E.31.1 Detailed Description ............................................................................ 105
  E.31.2 Constructor & Destructor Documentation ............................................. 105
    E.31.2.1 Size .................................................................................................. 105
  E.31.3 Member Data Documentation ................................................................ 105
    E.31.3.1 xSize .............................................................................................. 105
    E.31.3.2 ySize .............................................................................................. 106

E.32 FPVTE2012::StatusCode Class Reference .................................................. 106
  E.32.1 Member Enumeration Documentation ................................................ 106
    E.32.1.1 Kind ................................................................................................ 106

E.33 BiometricEvaluation::Error::StrategyError Class Reference ...................... 106
  E.33.1 Detailed Description ............................................................................ 107
  E.33.2 Constructor & Destructor Documentation ............................................. 107
    E.33.2.1 StrategyError ................................................................................... 107
    E.33.2.2 StrategyError ................................................................................... 107

Generated on July 3, 2012
Chapter 1

Introduction

1.1 Background

NIST has conducted several fingerprint-related evaluations in the last decade. The first fingerprint evaluation was called Proprietary Fingerprint Templates 2003 (PFT2003) and is now changed to PFTII. PFTII is a one-to-one matching evaluation that looks at the core match capabilities of the matching software. It does not evaluate one-to-many capabilities.

Minutiae exchange (MINEX), also a one-to-one matching evaluation, began in 2004. MINEX was started to support testing of fingerprint matching technologies using INCITS 378 standard interoperable templates. About a year later Ongoing MINEX was establish to support Personal Identity Verification (PIV).

The first one-to-many fingerprint evaluation conducted at NIST was FpVTE2003. FpVTE2003 had participants bring their own hardware and software to NIST for the evaluation and NIST supplied the data and retained all the matching results for final analysis. 1

1.2 Purpose

This document establishes the testing method for the one-to-many Fingerprint Vendor Technology Evaluation 2012 (FpVTE2012). In addition to describing the planned evaluation, this document includes the application program interface (API), software submission procedures, and participation application. See 2 for updates and additional documentation related to FpVTE2012.

FpVTE2003 only used enrollment sizes around 10,000 subjects. NIST has already conducted one-to-many biometric (face MBE2010 3 and iris IREXIII 4) evaluations using multi-million enrollment sizes in which all templates fit into the RAM of a single compute blade. FpVTE2012 will be the first biometric evaluation at NIST that partitions the enrollment set across multiple compute blades.

This technology evaluation is being done to meet several goals. FpVTE2012 will

- assess the current performance accuracy of one-to-many fingerprint matching software using operational fingerprint data,
- use enrolled sample sizes extending into the multiple millions,
- provide a testing framework and API for enrollment sizes that must be spread across the memory of multiple computer blades,
- support US Government and other sponsors in setting operational thresholds,
- evaluate on operational datasets containing newer datasets from live-scan ten-print “Identification Flats” capture systems, other live-scan capture devices (single finger and multi-finger), and historically significant scanned inked fingerprints. If available, datasets may include mobile device fingerprints.

1http://www.nist.gov/itl/iad/ig/fpvte03.cfm
2http://www.nist.gov/itl/iad/ig/fpvte2012.cfm
3http://www.nist.gov/itl/iad/ig/mbe.cfm
4http://www.nist.gov/itl/iad/ig/irex.cfm
1.3 Audience

Commercial, non-profit and research organizations with an ability to implement a large scale one-to-many fingerprint identification algorithm are invited to participate in FpVTE2012. Organizations only interested in one-to-one verification should participate in the Proprietary Fingerprint Testing II (PFTII).  

Participants will need to implement the API defined in this document. Participation is world wide and there is no charge. NIST expects to evaluate prototypical or experimental technologies.

1.4 Application Scenarios

The plan is for the submitted software to perform segmentation of all plain impression images or some alternative matching without segmentation. Planned testing scenarios for FpVTE2012 include the following:

1. 1 plain finger against enrolled database of 1 plain finger. The plain images are from single finger captures.
2. 2 plain fingers (right/left index) against enrolled database of 2 plain images. The plain images are from single finger captures.
3. 4, 8, 10 print Identification Flats (4-4-2, right/left four finger plain and two thumbs plain) against enrolled database of 10 print Identification Flats.
4. 10 rolled fingers against an enrolled database of 10 print rolled images.
5. 10 plain fingers against an enrolled database of 10 print plain images. Plain impression images will be 4-4-1-1, right/left four finger plain and right/left single thumb plain impression.
6. 10 plain fingers against an enrolled database of 10 print rolled images. Plain impression images will be 4-4-1-1, right/left four finger plain and right/left single thumb plain impression.

These scenarios will be split into three classes of participation. Class A will be the single finger capture scenarios (1 and 2). Class B will be the Identification Flats scenarios (3). Class C will be the 10 print rolled and slap scenarios (4-6).

All participants must submit successfully to the Class A scenario to participate in the other classes. Participants will send a different SDK for each class of participation. The three possible levels of participation are:

- Class A only.
- Classes A and B.
- Classes A, B, and C.

The next two tables show the enrollment and search sets that will be used in each class of participation. The last column of the enrollment sets show the various sizes of each enrollment set that will be tested.

---

5http://www.nist.gov/itl/iad/ig/pftii.cfm
## 1.4 Application Scenarios

### Enrollment Sets

<table>
<thead>
<tr>
<th>Set Name</th>
<th>Finger(s)</th>
<th>Data Type</th>
<th>Enrolled Sizes to be Tested</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1 (Class A)</td>
<td>1f Right Index</td>
<td>Plain Capture</td>
<td>5,000&lt;br&gt;10,000&lt;br&gt;100,000</td>
</tr>
<tr>
<td>E2 (Class A)</td>
<td>1f Left Index</td>
<td>Plain Capture</td>
<td>5,000&lt;br&gt;10,000&lt;br&gt;100,000</td>
</tr>
<tr>
<td>E3 (Class A)</td>
<td>2f Right and Left Index</td>
<td>Plain Capture</td>
<td>10,000&lt;br&gt;100,000&lt;br&gt;500,000&lt;br&gt;1,600,000</td>
</tr>
<tr>
<td>E4 (Class B)</td>
<td>10f Identification Flats (4-4-2)</td>
<td>Plain Capture</td>
<td>500,000&lt;br&gt;1,600,000&lt;br&gt;3,000,000</td>
</tr>
<tr>
<td>E5 (Class C)</td>
<td>10f</td>
<td>Rolled</td>
<td>500,000&lt;br&gt;1,600,000&lt;br&gt;3,000,000&lt;br&gt;5,000,000</td>
</tr>
<tr>
<td>E6 (Class C)</td>
<td>10f (4-4-1-1)</td>
<td>Plain Capture</td>
<td>500,000&lt;br&gt;1,600,000&lt;br&gt;3,000,000&lt;br&gt;5,000,000</td>
</tr>
</tbody>
</table>

### Search Sets

<table>
<thead>
<tr>
<th>Set Name</th>
<th>Finger(s)</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 (Class A)</td>
<td>1f Right Index</td>
<td>Plain Capture</td>
</tr>
<tr>
<td>S2 (Class A)</td>
<td>1f Left Index</td>
<td>Plain Capture</td>
</tr>
<tr>
<td>S3 (Class A)</td>
<td>2f Right and Left Index</td>
<td>Plain Capture</td>
</tr>
<tr>
<td>S4 (Class B)</td>
<td>4f Right Slap ID Flat</td>
<td>Plain Capture</td>
</tr>
<tr>
<td>S5 (Class B)</td>
<td>4f Left Slap ID Flat</td>
<td>Plain Capture</td>
</tr>
<tr>
<td>S6 (Class B)</td>
<td>8f Right and Left Slap ID Flat</td>
<td>Plain Capture</td>
</tr>
<tr>
<td>S7 (Class B)</td>
<td>10f ID Flats (4-4-2)</td>
<td>Plain Capture</td>
</tr>
<tr>
<td>S8 (Class C)</td>
<td>10f</td>
<td>Rolled</td>
</tr>
<tr>
<td>S9 (Class C)</td>
<td>10f (4-4-1-1)</td>
<td>Plain Capture</td>
</tr>
</tbody>
</table>

The next table shows which combination of search and enrollment sets will be tested in each class of participation.

### Testing for each Class of Participation

<table>
<thead>
<tr>
<th>Class</th>
<th>Search vs Enrollment Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>S1 x E1&lt;br&gt;S2 x E2&lt;br&gt;S3 x E3</td>
</tr>
<tr>
<td>B</td>
<td>S4 x E4&lt;br&gt;S5 x E4&lt;br&gt;S6 x E4&lt;br&gt;S7 x E4</td>
</tr>
<tr>
<td>C</td>
<td>S8 x E5&lt;br&gt;S9 x E5&lt;br&gt;S9 x E6</td>
</tr>
</tbody>
</table>
1.5 Timeline

The following timeline is planned for FpVTE2012.

- March - Release draft API for public comments.
- April 12 - Comments on Draft API due to NIST. Send to fpvte@nist.gov.
- April 30 - Release Second Draft API.
- May 11 - Comments on Second Draft API due to NIST. Send to fpvte@nist.gov.
- May 18 - Release Final API.
- June - Release driver software and validation data.
- June 29 - Deadline to submit application to participate.
- July 9 - Begin accepting SDKs for testing in Phase I.
- November 16 - End of Phase I testing.
- January 18 - Deadline to submit final SDK for testing in Phase II.
- April-May 2013 - Final Publication of results.

1.6 Offline Testing

While this test is intended to mimic operational reality, it remains an offline test performed on fixed sets of operational images maintained at NIST. The intent is to assess the core capability of the algorithms. The offline testing does allow for uniform, fair, repeatable, and efficient evaluation of underlying technologies. Testing under a fixed API allows for a detailed set of performance related parameters to be measured.

1.7 Phased Testing

There are two Phases for submitting software to NIST for testing. To support research and development and to get robust and complete assessment of the algorithmic capabilities, in Phase I FpVTE will be conducted over several rounds. These test rounds are intended to support development of improved recognition performance. Once the test commences, NIST will accept implementations on a first-come-first-served basis and will return results to participants as expeditiously as possible. Participants may submit revised implementations to NIST only after NIST provides results for the prior submission. The frequency with which a participant may submit implementations to NIST will depend on the times needed for participant preparation, validation, execution, scoring, and participant review and decision processes.

There will be a maximum number of SDKs allowed for submission from each participant during phase I but that number will be based partially on the total number of participants. It will be communicated to actual participants after the final application deadline.

Once Phase I completes participants will have about 45 days to make changes and submit a final SDK (per class of participation) at the start of Phase II. Phase II will not be done in rounds and results will only be released in the Final FpVTE report.

Participants must submit software in Phase I in order to participate in Phase II.

1.7.1 Interim Reports

Each submission will result in a “score-card” provided to the participant. While the score-card may be used by the participant for arbitrary purposes, they are intended to promote development. The score-cards may include:
1.8 Reporting of Failure to Enroll or Acquire (FTE, FTA)

- machine generated (i.e. scripted),
- provided to participants with identification of their algorithm,
- include template size, timing, and performance accuracy results (i.e. DET, ROC, etc.),
- include anonymous results from other participant implementations,
- are regenerated as new implementations become available and when new analysis is added.

NIST does not intend to release these test reports but may share such information with U.S. Government test sponsors without releasing the participant names.

1.7.2 Publication of results

Once NIST completes the testing rounds, one or more final reports will be released.

- A NIST Interagency Report (NISTIR) will be published.
- NIST may publish/present results in other academic literature, conferences, or workshops.

The final report will publish results for all submissions, with the intent being to show progress and performance trade-offs made by the implementations. Ultimately, the reported results will highlight the most capable implementations.

IMPORTANT: Results will be attributed to the providers in the final reporting.

1.8 Reporting of Failure to Enroll or Acquire (FTE, FTA)

FTE and FTA have different meanings in production systems where humans interact with biometric readers. For offline testing, soft failures, where the algorithm elects not to produce a template (e.g. image quality reasons) shall be treated identical to failures, where the software crashes or hangs. For this test, any failure to convert images into templates shall be counted as a FTE and reported as such. Further instructions on handling FTE cases are provided in the next section.

1.9 Matching of Empty, Broken, and Missing Templates

For a FTE, the template generator must return an empty (0 byte) template.

After software failure, the absence of an output template will cause NIST to produce an empty (0 byte) template.

For enrollment templates the finalization step must handle these empty templates.

NIST will not call the one-to-many search function if the preceding search template is empty (i.e. FTE). The search template will be stored with a candidate list of all -1 match scores.

1.10 Reporting of Template Size

Because template size is influential to storage requirements, computational efficiency, and wireless/mobile applications, this API supports measurement of template size. NIST will report statistics on the actual size of the templates produced by implementations submitted to this test.
1.11 Runtime Memory Usage

NIST will attempt to allocate enough blades such that the enrollment data can be distributed across them without enrollment data storage exceeding 75% of the blade’s memory. The other 25% will be reserved for OS and implementation usage. The blades being used for the evaluation have 192GB of memory. Based on these specifications there will be 144GB available for enrollment data and 48GB for OS and implementation usage. NIST may report the amount of memory used during one-to-many searches.

1.12 Reporting of Computational Efficiency

As with other tests, NIST will compute and report performance accuracy. Additionally, NIST will report timing statistics for core functions of the submitted implementation. This includes feature extraction and identification functions.

1.13 Exploring the Accuracy-Speed Trade-Off

Each participant will be allowed/encouraged to submit two implementations for evaluation. The intent of this decision is to demonstrate the trade off between accuracy and speed, but participants are not required to submit "fast" and "slow" variants. Participants may, for example, choose to submit a mature and an experimental implementation.

NIST will be reporting the performance of single and two finger (Class A) separately for slow and fast submissions. The intent is to report fast SDKs seen as more viable for positive access control application and highlight not just the performance of the algorithm but also the speed. "Fast" SDKs will be those that can do a single search vs. the 100K enrollment set in under 20 seconds and "slow" will be anything over that search time.

1.14 Hardware Specification

NIST intends to support high performance by specifying the runtime hardware beforehand. There are several types of computer blades that may be used in the testing. The blades are labeled as Dell M905, M910, M605, and M610. The M610, M905 and M910 have 192GB of total RAM available per blade. The M605 blades have 64GB of total memory available but will only be used for template generation. The following list gives more details about the hardware of each blade type:

- Dell M605 - Dual Intel Xeon E5405 2 GHz CPUs (4 cores each)
- Dell M610 - Dual Intel Xeon X5680 3.3 GHz CPUs (6 cores each)
- Dell M905 - Quad AMD Opteron 8376HE 2 GHz CPUs (4 cores each)
- Dell M910 - Dual Intel Xeon X7560 2.3 GHz CPUs (8 cores each)

NIST is requiring the use of 64-bit architecture. This will support large memory allocation to support 1:N identification with image counts in the multi-millions. Note that while the API allows read access of the disk during execution, disk access may be prohibitively slow.

Some of the API calls allow the implementation to write persistent data to hard disk. Note that the file system used doesn’t efficiently support creation of millions of small files. Participants are strongly encouraged to use some database structure to store the finalized enrollment templates.

1.15 Operating system and Compilation Environment

All submitted implementations will be tested on 64-bit CentOS 6.2 (2.6.32-220.7.1.el6.x86_64). NIST will link the provided libraries to our C++ language test driver. Participants are required to provide their library in a format that is linkable using GNU g++ 4.4.6-3. The standard libraries are:
1.16 Threaded Computations

Threaded computations will be allowed for finalizing the enrollment set only. All other functions are not to perform multi-threaded computations for this test. For these other functions, the NIST driver will be running multiple independent instances.

1.17 Time Limits

The reported template extraction timing statistics will be based on a subset on the data run on a dedicated compute blade. These time limits are per image. If all ten fingers of a subject are being extracted, the time limit is increased by a factor of 10.

- Feature extraction - should be accomplished in 3 seconds or less for each input fingerprint. A four finger slap counts as four input fingerprints.
- Finalization - should be accomplished in 12 hours or less on a single compute blade using multiple cores. The finalization blade will have 2 CPU with 6 core each.
- Identification/Search - See the table below for information on search timings. Search times will be averaged over the search set and this average should not exceed the times listed in the table below. Note that for Class A (1f and 2f) that results will be reported separately for "fast" algorithms (<20s) and "slow" algorithms (>20s). A single search time is the total combined time measured for stage one and stage two identification (not including initialization times). This will include the I/O time in stage one identification needed to output the data needed for stage two identification. The "Stage 1 Data Directory" will be RAM Disk storage that will be moved to hard drive storage by the test driver. This data will be moved back to RAM disk storage (by the test driver) for use in stage two identification.

<table>
<thead>
<tr>
<th>Class</th>
<th>Search vs Enrollment Set</th>
<th>Time per Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>S1 x E1</td>
<td>&lt;20s (fast)</td>
</tr>
<tr>
<td></td>
<td>S2 x E2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S3 x E3</td>
<td>&lt;500s (slow)</td>
</tr>
<tr>
<td></td>
<td>S4 x E4</td>
<td>&lt;90s</td>
</tr>
<tr>
<td></td>
<td>S5 x E4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S6 x E4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S7 x E4</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>S8 x E5</td>
<td>&lt;90s</td>
</tr>
<tr>
<td></td>
<td>S9 x E5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>S9 x E6</td>
<td></td>
</tr>
</tbody>
</table>

1.18 Ground Truth Integrity

The test datasets are derived from operational systems. They may contain ground truth errors in which:
• a single subject is present under multiple different identifiers,
• two subjects are present under one identifier, or
• finger positions are mislabeled.

If these errors are detected, they will be removed or repaired. NIST will use aberrant scores (high impostor scores/low genuine scores) to detect such errors. This process will be imperfect, and residual errors are likely to remain. For comparative testing, identical datasets will be used and the presence of errors should give an additive increment to all error rates. For very accurate implementations this could dominate the error rate. NIST intends to attach appropriate caveats to the accuracy results. For prediction of operational performance, the presence of errors gives incorrect estimates of performance.
Chapter 2

Datasets

This section describes the datasets that will be used for the evaluation.

2.1 Image Types

The evaluation dataset used in FpVTE2012 will be from operational datasets made available to NIST for fingerprint evaluations. The datasets are government use only and will not be released to the public. The datasets will, to the extent permitted by law, be protected under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a) as applicable.

The datasets will be comprised of several fingerprint impression types, including rolled, multi-finger plains, and single finger plains. Rolled are all individual captures that attempt to capture the full width of the fingerprint by rolling from side to side during capture. Multi-finger plains capture the four left/right fingers all at the same time and for Identification Flats (IDFlats) the two thumbs are also captured at the same time. Single finger plains are individual captures of the left and right index fingers on a single finger capture device. If available, the single finger captures may include fingerprints captured on mobile devices. FpVTE2012 will look at performance of plain-to-rolled, plain-to-plain, and if time and resources permit rolled-to-rolled.

Many of the datasets are larger samples of data used in previous NIST evaluations such as PFT, MINEX, NFIQ, and FpVTE2003. The single finger capture fingerprint images will be data from DHS. The ten print rolled and slap (4-4-1-1) fingerprint images will be data from FBI, DHS, LACNTY, AZDPS, and TXDPS. The Identification Flat (4-4-2) fingerprint images will be from DHS.

All images in the datasets are 8-bit grayscale. Images have been compressed using Wavelet Scalar Quantization (WSQ) compression, but they will be passed to the submitted implementation as uncompressed raw pixel images. All images were scanned at 500 pixels per inch and will have varying dimensions that will be passed in a data structure to the implementation.

Multiple finger plain captures will not be segmented into individual fingerprints. The implementation is required to perform any needed segmentation of the fingerprints.
Chapter 3

SDK Application Programming Interface

This section defines the interface for software submitted to the Fingerprint Vendor Technology Evaluation 2012. FpVTE2012 is intended to be a large scale one-to-many identification evaluation of proprietary fingerprint templates.

FpVTE2012 will only use CentOS 6.2 (2.6.32-220.7.1.el6.x86_64), no other operating systems are allowed. All software must run under CentOS 6.2 (2.6.32-220.7.1.el6.x86_64).

The intent of this API is to support:

- distribution of the enrolled templates across the memory of multiple blades,
- distributed enrollment and identification with multiple instance of each process running independently and in parallel,
- recovery after a fatal exception and record the number of failures,
- the “black-box” nature of biometric templates and matching,
- the freedom of the provider to use arbitrary algorithms,
- measure duration of core function calls,
- measure template sizes,
- measure matching performance.

The following sections give general descriptions of the processes that make up the API and their role in the evaluation. Details of the subroutine interface are given in section 3.5 and Appendix D.

3.1 1:N Identification with Large Enrollment Set

The 1:N identification has two main phases, enrollment and identification. Enrollment has two steps. First all the templates used in the enrolled set are extracted from the fingerprint images. Second, a process is run that finalizes the enrollment templates used during identification. Identification is completed in three steps. First, the templates are extracted for the search image(s). Second, the extracted search templates are passed to stage one of identification, which searches the entire set of enrolled templates, across several blades, for potential matches. Finally, stage two identification is invoked and uses the results from stage one identification to produce the final candidate list of the top 100 potential matches to the search subject.

The expectation is that the total memory needed to store the entire set of enrolled templates in RAM will exceed the memory capacity of a single blade available for the evaluation. For this reason, stage one identification is accomplished by dividing the enrolled templates into pieces, where each piece can be loaded into RAM on separate blades. Stage one identification searches each piece of the divided enrollment set separately and stores the needed information for that search, so stage two of identification can produce a candidate list of the top 100 potential matches to the search subject.
3.2 Input Fingerprint Image Structure

While the expectation is dividing the enrollment set into pieces across blades, it is conceivable for a given test within the evaluation that the enrolled templates could fit into the memory of a single blade. Therefore, implementations should be able to handle any number of blades being used for stage one identification ($1 \leq B \leq B_{total}$). Additionally, it is expected that if a test were run with $B = 1$ and the same test were run with $B = 2$ (or more) the resulting candidate list would be the same.

Each function in the API enrollment process, search template extraction, stage one identification, and stage two identification call initialization routines that allow the implementation to pre-configure itself as needed. No additional initialization functions are allowed and all configuration information will be located in the configuration directory which has read only access.

### 3.1.1 Threading and Multi-Processing

Enrollment finalization is the only function allowed to use multi-threading. It is a single process that must run to completion before proceeding to the identification stages.

For all other functions (template extraction and identification) the NIST driver will handle invoking multiple processes to fully utilize the available resources. Implementations submitted should anticipate that each stage, enrollment/search template extraction and stage one/two identification will have multiple instances invoked by the NIST driver. These instances could be across multiple CPUs/cores on a single blade or across multiple blades. It is important that the implementation account for this during process initialization so that each instance runs completely independent of the others.

This multiple instance implementation was used in a previous 1-to-many evaluation (IREXIII) and multiple searches were performed against an enrollment set in the memory of a single blade as long as the memory was not changed during the search. FpVTE 2012 will use this same plan to load the enrollment templates into memory. Then using the multiple cores available on the blades, the test driver can invoke multiple searches against that enrolled set at the same time. Again, the key is that the enrolled set loaded into the blade memory must remain unchanged/static during searching.

### 3.1.2 Implementation Identifiers

Each implementation must return a NIST assigned ID and a contact email address.

### 3.2 Input Fingerprint Image Structure

Structure details are in Appendix D (D.2).

FpVTE2012 will enroll multiple fingerprints from a subject in a single function call. The input data structure *FingerImage* will be used for passing images for a single subject into the template extraction process. The details related to this structure are described in the reference documentation *FingerImage* and *FingerImageSet*. *FingerImage* defines the structure for a single fingerprint image and *FingerImageSet* is a vector of *FingerImage* for each finger of that subject. If the test is using 10 rolled fingerprints for each subject the *FingerImageSet* for each subject will contain *FingerImage* s. If the test is 10 plain fingerprint images per subject the input *FingerImageSet* will have *FingerImage(s)* for the left/right four finger plains and left/right thumb plains. Any segmentation into individual fingerprints is left to the implementation.

The number of images per subject will vary depending on the specific test being performed, but will typically be either 1, 2, 4, or 10 fingerprint images. The implementation will need to support this accordingly in enrollment and matching.

### 3.3 Output Fingerprint Template Structure

Structure details are in Appendix D (D.3).

The template extraction process will return both the resulting fingerprint templates and a data structure *FingerOutput* containing information about the templates (i.e. actual template size) and the input fingerprints for a given subject. The format of the returned templates are uncontrolled by FpVTE2012. Details related to the structure can be found in the reference documentation *FingerOutput* and *FingerOutput*. 

Generated on July 3, 2012
3.3 Output Fingerprint Template Structure

3.3.1 Enrollment Template Extraction and Finalization

The enrollment template extraction process is accomplished in two steps. First, templates for each subject in the enrolled set are extracted independently of each other. This allows for multiple processes to run simultaneously across many cores/ blades. After all templates are extracted, the second step is the enrollment finalization described in more detail in section 3.3.1.2.

3.3.1.1 Enrollment Template Extraction

Function call parameters and details are in Appendix D (D.5.1.2 and D.5.1.3).

The enrollment template extraction process is shown in figures 3.1 and 3.2. For each instance of the enrollment template extraction process, the NIST driver will call the enrollment initialization function once before multiple calls to enrollment template extraction. The resulting templates are then stored in a template RecordStore. The RecordStore is described in section 3.5.3.1. The enrollment template extraction process will be accomplished by running multiple template extraction processes in parallel across multiple cores and blades. NIST will merge the multiple template RecordStores created into a single enrollment template RecordStore that gets passed to the enrollment finalization process.

3.3.1.2 Enrollment Finalization

Function call parameters and details are in Appendix D (D.5.1.4).

After extracting all enrollment templates, the enrollment finalization process shown in figures 3.3 and 3.4 is called. The input to this process is the merged single enrollment RecordStore and the output is a directory that the application will populate as needed. The implementation must divide the finalized enrollment set into B partitions, one for each of the blades in the search process. The NIST test driver will tell the finalization process how many blades (B) can be used for identification. After finalization, the enrollment set is “frozen” and this directory will be made read only during template searching.
Finalization allows the implementation to conduct, for example, statistical processing of the feature data, indexing, and data reorganization. The function may alter the file structure. It may increase or decrease the size of the stored data. No output to the test driver is expected from this function except a successful StatusCode.

The format of the enrollment directory is completely controlled by the participant. A major note here is that the file system used in the NIST test environment is not optimal for large numbers of small files. If an implementation is creating a large number of individual files, it will impact performance and most likely be excluded from the evaluation. Implementations must not create individual files for each subject in the enrollment set.

### 3.3.2 Search Template Extraction

Function call parameters and details are in Appendix D (D.5.1.5 and D.5.1.6).

This section of the API describes the search template extraction process as shown in figures 3.5 and 3.6. The enrollment templates will not be used as search templates. This allows for a different function to be used for search template creation. Like enrollment template creation, the NIST driver will call initialization once for each instance of search template creation before multiple calls to search template creation. The resulting templates will be stored for later testing or passed directly to stage one identification.
3.3 Output Fingerprint Template Structure

Figure 3.4: Calling sequence for enrollment template finalization process

Figure 3.5: Flowchart of search feature template extraction process

Figure 3.6: Calling sequence for search feature template extraction process
### 3.3.3 Identification Stage One

**Function call parameters and details are in Appendix D (D.5.1.7 and D.5.1.8).**

This section describes the first stage of identification as shown in figures 3.7 and 3.8. For this stage of identification, the enrollment set will be stored in pieces across multiple blades. The number of blades (B) will be the same as was input for enrollment finalization. Each separate piece of the enrollment set is loaded into memory by the implementation during the initialization step. The NIST driver will run multiple search subjects against this piece of the enrollment set simultaneously on that blade.

There will be a 4 GB storage area in which results from this first stage of matching (Stage 1 Data Directory) can be stored for each search subject. This storage area will be a RAM disk and is shared by all identification processes running on that blade. In an effort to reduce disk I/O from match timing functions, the moving of stage one identification results data from the dedicated storage area to a shared SAN storage device is done by the NIST driver after the call to stage one identification is completed for each search subject.

Once results are moved to the SAN storage, they are removed from the dedicated storage freeing up space for other stage one results. It is important to note that the output directory for stage one identification results is shared by all B blades running stage one identification processes. This directory is where the output results from all stage one identification processes are being grouped back together for processing in stage two identification. The multiple identification processes in stage one across the blades are asynchronous, but stage two identification can not be run until all blades have finished stage one for a specific search subject.

### 3.4 Output Candidate Lists Structure

**Structure details are in Appendix D (D.1).**

The second identification stage will produce a candidate list for results analysis. The list will be returned in a data structure defined in the reference documentation `Candidate` and `CandidateSet`.

#### 3.4.1 Identification Stage Two

**Function call parameters and details are in Appendix D (D.5.1.9 and D.5.1.10).**
This section describes the second stage of identification as shown in figures 3.9 and 3.10. After initializing this stage of identification, multiple search subjects will be passed to the identification process, one at a time. The search subjects will be the same ones from stage one identification. The second stage of identification will have access to both the original enrollment directory and the data directory containing the output of stage one identification. The stage one output will be reloaded into a RAM disk for use in stage two identification. Stage two identification will output the candidate list and required scoring data for the search subject.

There is no intended control on how stage two reaches the final candidate list but the search time limit that combines stage one and stage two search times must be followed. Stage two could be as simple as sorting/organizing stage one results to get the candidate list or it could involve a more complex algorithm that involves some level of additional matching of feature templates.

3.5 SDK C++ Interface

FpVTE has defined an abstract C++ [1] class that must be implemented by the software under test. This class, SDKInterface, contains the methods that will be called by the NIST SDK driver software. SDK vendors must deliver a library that implements the abstract interface defined by that class. C++ was chosen in order to make use of some object-oriented features and for interoperability with existing C code.

3.5.1 The SDK Interface

The abstract class SDKInterface must be implemented by the SDK vendor. The processing that takes place during each phase of the test is done via calls to the methods declared in the interface as pure virtual, and therefore are to be implemented by the SDK. The test driver will call these methods, handling all return values and exceptions. Example code will be distributed by NIST to show how an SDK can implement the methods.

An example of an implementation’s header file might be:

```
#include <fpvte2012.h>

namespace FPVTE2012 {

class NullSDK : public FPVTE2012::SDKInterface {

public:

    NullSDK();
    ~NullSDK();

    void getIDs()
};
}
```
3.5 SDK C++ Interface

Figure 3.9: Flowchart of second stage of identification process

Figure 3.10: Calling sequence for second stage of identification process
Listing 3.1 shows how to subclass the abstract interface class and declare the implementation of the `getIDs()` method. The definition of the method is then:

```cpp
# include <fpvteNullSDK.h>

using namespace BiometricEvaluation;
using namespace FPVTE2012;

NullSDK::NullSDK () { }
NullSDK::~NullSDK() { }

void
NullSDK::getIDs(
    string &nist_assigned_identifier,
    string &email_address)
{
    nist_assigned_identifier = "0x12345678";
    email_address = "foo@example.com";
}

SDKIptr
SDKInterface::getSDK()
{
    NullSDK *p = new NullSDK();
    SDKIptr ap(p);
    return (ap);
}
```

The other methods of the SDKInterface class are implemented in a similar manner.

There is one class (static) method declared in `SDKInterface`, `getSDK()` which must also be implemented by the SDK. This method returns a shared pointer to the object of the interface type, an instantiation of the SDK implementation class. A typical implementation of this method is also shown in Listing 3.2.

### 3.5.2 Exceptions

Many of the SDK API methods are declared to throw an exception in the case of error. SDKs should only throw an exception when the method call cannot be completed. For example, in the matching call, the SDK should *not* throw an exception if there is a failure to match, or the image quality is too low. An example of the appropriate use of the exception is when a memory allocation failure, or an input template is corrupted. The exception string should then be filled out with as much detail as desired to enable debugging.

### 3.5.3 Biometric Evaluation Classes

The API makes use of several classes that are part of the NIST Image Group Evaluation framework. The pertinent components of the framework will be provided to SDK vendors as source code, including a build system for the Linux operating system. Descriptions of the framework classes are provided in this section.
3.5.3.1 IO::RecordStore

A RecordStore is an abstraction that associates keys with an opaque object. These key-value pairs are unique, based on the key which is a string. Applications, in addition to retrieving the data associated with a key, can insert, remove, or replace the data. Functionality is also provided to sequence through a RecordStore; however, there is no guarantee that the key-value pairs will be retrieved in the order they were inserted.

The API defined for FpVTE provides an opened RecordStore to the SDK as read-only in some cases. The SDK must not attempt to insert or remove a key for those RecordStores else an exception will be thrown.

For FpVTE2012 the RecordStore is being used to store the large set of enrollment templates in a single file that can then be passed to the finalization process.

3.5.3.2 Memory::AutoArray

The AutoArray is used to manage an array of objects in a safe manner with the efficiency associated with standard C++ arrays. The size of the array can be changed dynamically, usually done when a larger array is required.

Many of the SDK interface functions pass a buffer containing an image or a biometric template. This buffer is defined as a uint8Array, which is an AutoArray of unsigned 8-bit integers.

3.5.3.3 Error::Exception

All exceptions thrown by the SDK are of the Exception class. An object of this class can be constructed with a string giving more details on the nature of the exception. The functions implemented by the SDK should set the exception string to contain information that will be useful for debugging.

Some of the methods of the framework objects throw exceptions which must be handled by the SDK. All of these exception objects are subclasses of the Exception class. Therefore, the SDK can simply catch Exception, and then may simply rethrow it, or throw a new Exception object with more detailed information in the information string of the Exception object. In all cases the SDK methods must throw an Exception object.
Chapter 4

Software and Documentation

This section describes the specific software and documentation requirements for implementations submitted by participants.

4.1 Library and Platform Requirements

Participants shall provide NIST with binary code only (no source code). Header files shouldn’t be necessary, but they are allowed. If used, they shall not contain intellectual property of the participant nor material that is otherwise proprietary. It is preferred the implementation be submitted in the form of a single static library. However, dynamic and shared library files are permitted.

The core library shall be named as follows:

```
libFpVTE12_NIST_Assigned_ID_[A, B, or C]_[1 or 2]_sequence.[so or a]
```

* NIST_Assigned_ID  The implementation identifier assigned by the NIST test liaison.
* A, B, or C  Class of participation for this submission.
* 1 or 2  “Fast” (1) or “slow” (2) submission for this implementation.
* sequence  Two digit sequence number, incremented by 1 when a new implementation is sent to NIST (i.e. 01, 02, …).

If necessary, additional dynamic or shared library files may be submitted that support this “core” library file. The core library file may have dependencies implemented in these other files.

The use of any graphic processing units (GPU) is not permitted.

4.2 Configuration and Vendor-Defined Data

The implementation under test may be supplied with configuration and supporting data file in the configuration directory. NIST will report the size of the supplied configuration files.

4.3 Linking

NIST will link the provided library file(s) to a C++ language test driver application developed at NIST. The runtime environment will be CentOS 6.2 (2.6.32-220.7.1.el6.x86_64), with GNU g++ 4.4.6-3 (PREFERRED).
The NIST test driver is compiled using GNU g++ 4.4.6-3 and participants are required to provide their library in a format that is linkable to the NIST test driver using GNU g++ 4.4.6-3. Thus, participants are strongly advised to verify library-level compatibility with GNU g++ 4.4.6-3 (on an equivalent platform) prior to submitting their software to NIST. This will help avoid linkage problems once the software is sent to NIST.

FpVTE2012 will only use CentOS 6.2 (2.6.32-220.7.1.el6.x86_64), no other operating system is allowed. All software must run under CentOS 6.2 (2.6.32-220.7.1.el6.x86_64). A copy of the ISO used to build the NIST systems is available at http://nigos.nist.gov:8080/evaluations/CentOS-6.2x86_64binDVD1.iso

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

### 4.4 Installation and Usage

The implementation must install easily (no participant interaction required) to be tested, and shall be executable on any number of machines without additional machine-specific license control procedures or activation. The implementation shall be installable using simple file copy methods and shall not require the use of a separate installation program.

The implementation shall neither implement nor enforce any usage controls or limits based on licenses, number of executions, presence of temporary files, etc. The implementation shall remain operable until December 31, 2013.

Hardware activation dongles are not acceptable.

### 4.5 Modes of Operation

Individual implementations provided shall not include multiple “modes” of operation, or algorithm variations. No switches or options are allowed within a library. For example, the use of two different “coders” by a feature extractor must be split across two separate implementation library submissions.

### 4.6 Runtime Behavior

#### 4.6.1 Interactive Behavior

The implementation will be run in a non-interactive batch mode, without terminal support. Therefore, the submitted library shall not use any interactive functions such as graphical user interface (GUI) calls or any other calls which require any type of terminal interaction.

#### 4.6.2 Status Messages

Since the implementation will be run in a non-interactive batch mode, the submitted library must run quietly meaning no messages should be written to standard output or error. An implementation may write debugging messages to a log file, which must be declared in the documentation.

#### 4.6.3 Error Codes and Exception Handling

The implementation should report errors back the the caller by setting the appropriate status code, and optionally providing an information string. Section 3.5 describes the handling of exceptions encountered by the SDK. In the case of an exception being thrown by the SDK, the returned status object will not be available to the caller. Therefore, use of the information string inside the exception object is encouraged.
4.6 Runtime Behavior

4.6.4 External Communication

Processes running on NIST hosts shall not create side effects in 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, network, or other process).

Implementations shall not attempt to read any resource other than those explicitly allowed in this document. If detected, NIST reserves the right to cease evaluation of all implementations from the participant, notify the participant, and document the activity in the published reports.

4.6.5 Stateful Behavior

All components in this test shall be stateless and deterministic, except as noted. This applies to image segmentation, template creation and matching. Thus, all functions should give identical output, for a given input, independent of the runtime history. This holds for enrollment set partitioning. For example, results should be the same if the enrollment set is spread across four blades or eight blades. NIST will institute appropriate tests to detect stateful behavior. If detected, NIST reserves the right to cease evaluation of all implementations from the participant, notify the participant, and document the activity in the final reports.
Chapter 5

How to Submit Implementations to FpVTE2012

5.1 Confidentiality and Integrity Protection

NIST requires that all software, data, and configuration files submitted by the participants be signed and encrypted. Signing is done with the participants private key, and encryption is done with the NIST public key. The detailed commands for signing and encrypting are given here:

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 submission, we ensure authenticity (the software actually came from the submitter.) **NIST will not accept 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.**

5.2 How to Participate

Those wishing to participate in FpVTE2012 must meet all the deadlines of the FpVTE 2012 calendar.

Software must be submitted as follows:

- **IMPORTANT:** Follow the instructions for encrypting submissions in section ??.
- Complete the “Application to Participate in FpVTE2012”.
- Provide an implementation library (encrypted) that complies with the API specified in this document.
  - Encrypted data and implementations below 20MB can be emailed to NIST at fpvte@nist.gov
  - Encrypted data and implementations over 20MB can be
    - Made available for download from generic websites where NIST is not required to register or establish any membership, or
    - FEDEX/UPS or similar shipping method (so package can be tracked) to NIST on CD/DVD at this address (tracking number must be emailed to NIST):

    NIST
    c/o FpVTE2012 Test Liaison
    100 Bureau Drive MS 8940
    Gaithersburg, MD 20899-8940
    USA
5.3 Implementation Validation

Participants will be provided validation datasets containing imagery from publicly available datasets in the same format as the evaluation test datasets. These validation datasets will be used to test implementation functionality before submitting to NIST. The output from these validation datasets must be submitted with the implementation. NIST will repeat the processing of the validation datasets on our hardware to confirm the implementation is functioning correctly.
Bibliography

# Appendix A

## Namespace Index

### A.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<table>
<thead>
<tr>
<th>Namespace Path</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiometricEvaluation::Error</td>
<td>Exceptions, and other error handling</td>
<td>30</td>
</tr>
<tr>
<td>BiometricEvaluation::Finger</td>
<td>Biometric information relating to finger images and derived information</td>
<td>31</td>
</tr>
<tr>
<td>BiometricEvaluation::Image</td>
<td>Basic information relating to images</td>
<td>32</td>
</tr>
<tr>
<td>BiometricEvaluation::IO</td>
<td>Input/Output functionality</td>
<td>33</td>
</tr>
<tr>
<td>BiometricEvaluation::IO::Utility</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>BiometricEvaluation::Text</td>
<td>Text processing for string objects</td>
<td>40</td>
</tr>
<tr>
<td>FPVTE2012</td>
<td>Contains all the structures and functions used by the API</td>
<td>40</td>
</tr>
</tbody>
</table>
Appendix B

Class Index

B.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BiometricEvaluation::Memory::AutoArray<T> .................................................. 43
FPVTE2012::Candidate ...................................................................................... 49
BiometricEvaluation::Image::CompressionAlgorithm ........................................... 50
BiometricEvaluation::Image::Coordinate ............................................................ 51
BiometricEvaluation::Error::Exception ............................................................. 53
  BiometricEvaluation::Error::ConversionError .................................................. 50
  BiometricEvaluation::Error::DataError .......................................................... 52
  BiometricEvaluation::Error::FileError ........................................................... 55
  BiometricEvaluation::Error::MemoryError ..................................................... 68
  BiometricEvaluation::Error::NotImplemented ............................................... 69
  BiometricEvaluation::Error::ObjectDoesNotExist .......................................... 70
  BiometricEvaluation::Error::ObjectExists ..................................................... 71
  BiometricEvaluation::Error::ObjectIsClosed ............................................... 72
  BiometricEvaluation::Error::ObjectIsOpen .................................................... 73
  BiometricEvaluation::Error::ParameterError ................................................. 73
  BiometricEvaluation::Error::StrategyError ................................................... 106
FPVTE2012::FingerImage ................................................................................... 61
BiometricEvaluation::Finger::FingerImageCode ............................................... 61
FPVTE2012::FingerOutput .................................................................................. 62
BiometricEvaluation::Image::Image .................................................................. 63
  BiometricEvaluation::Image::Raw ................................................................. 83
  BiometricEvaluation::Finger::Impression ....................................................... 68
  BiometricEvaluation::Finger::PatternClassification ....................................... 74
  BiometricEvaluation::Finger::Position ........................................................... 75
  BiometricEvaluation::IO::Properties ............................................................. 75
  BiometricEvaluation::IO::PropertiesFile ....................................................... 81
BiometricEvaluation::IO::RecordStore ............................................................ 84
  BiometricEvaluation::IO::FileRecordStore .................................................... 56
BiometricEvaluation::Image::Resolution .......................................................... 97
FPVTE2012::ReturnStatus .................................................................................. 98
FPVTE2012::SDKInterface ............................................................................... 99
BiometricEvaluation::Image::Size .................................................................... 105
FPVTE2012::StatusCode .................................................................................. 106
Appendix C

Class Index

C.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- **BiometricEvaluation::Memory::AutoArray<T>**  
  A C-style array wrapped in the facade of a C++ STL container  
  43

- **FPVTE2012::Candidate**  
  Object defining format of candidate reporting  
  49

- **BiometricEvaluation::Image::CompressionAlgorithm**  
  Image compression algorithms  
  50

- **BiometricEvaluation::Error::ConversionError**  
  Error when converting one object into another, a property value from string to int, for example  
  50

- **BiometricEvaluation::Image::Coordinate**  
  A structure to contain a two-dimensional coordinate without a specified origin  
  51

- **BiometricEvaluation::Error::DataError**  
  Error when reading data from an external source  
  52

- **BiometricEvaluation::Error::Exception**  
  The parent class of all BiometricEvaluation exceptions  
  53

- **BiometricEvaluation::Error::FileError**  
  File error when opening, reading, writing, etc  
  55

- **BiometricEvaluation::IO::FileRecordStore**  
  56

- **FPVTE2012::FingerImage**  
  Object containing the fingerprint image and corresponding attributes  
  61

- **BiometricEvaluation::Finger::FingerImageCode**  
  61

- **FPVTE2012::FingerOutput**  
  Object for output data related to an extracted template  
  62

- **BiometricEvaluation::Image::Image**  
  Represent attributes common to all images  
  63

- **BiometricEvaluation::Finger::Impression**  
  Finger and palm impression types  
  68

- **BiometricEvaluation::Error::MemoryError**  
  An error occurred when allocating an object  
  68

- **BiometricEvaluation::Error::NotImplemented**  
  A NotImplemented object is thrown when the underlying implementation of this interface has not or could not be created  
  69

- **BiometricEvaluation::Error::ObjectDoesNotExist**  
  The named object does not exist  
  70
BiometricEvaluation::Error::ObjectExists
   The named object exists and will not be replaced ............................................. 71

BiometricEvaluation::Error::ObjectIsClosed
   The object is closed ............................................................................................. 72

BiometricEvaluation::Error::ObjectIsOpen
   The object is already opened .................................................................................. 73

BiometricEvaluation::Error::ParameterError
   An invalid parameter was passed to a constructor or method ..................................... 73

BiometricEvaluation::Finger::PatternClassification
   Pattern classification codes ....................................................................................... 74

BiometricEvaluation::Finger::Position
   Finger position codes ............................................................................................... 75

BiometricEvaluation::IO::Properties
   Maintain key/value pairs of strings, with each property matched to one value .......... 75

BiometricEvaluation::IO::PropertiesFile
   A Properties object persisted in an file on disk ....................................................... 81

BiometricEvaluation::Image::Raw
   An image with no encoding or compression ............................................................... 83

BiometricEvaluation::IO::RecordStore
   A class to represent a data storage mechanism .......................................................... 84

BiometricEvaluation::Image::Resolution
   A structure to represent the resolution of an image .................................................... 97

FPVTE2012::ReturnStatus
   A structure to contain information about a failure by the software under test ............ 98

FPVTE2012::SDKInterface .......................................................................................... 99

BiometricEvaluation::Image::Size
   A structure to represent the size of an image, in pixels ............................................. 105

FPVTE2012::StatusCode ........................................................................................... 106

BiometricEvaluation::Error::StrategyError
   A StrategyError object is thrown when the underlying implementation of this interface encounters an error ......................................................... 106
Appendix D

Namespace Documentation

D.1 BiometricEvaluation::Error Namespace Reference

Exceptions, and other error handling.

Classes

• class Exception
  The parent class of all BiometricEvaluation exceptions.
• class FileError
  File error when opening, reading, writing, etc.
• class ParameterError
  An invalid parameter was passed to a constructor or method.
• class ConversionError
  Error when converting one object into another, a property value from string to int, for example.
• class DataError
  Error when reading data from an external source.
• class MemoryError
  An error occurred when allocating an object.
• class ObjectExists
  The named object exists and will not be replaced.
• class ObjectDoesNotExist
  The named object does not exist.
• class ObjectIsOpen
  The object is already opened.
• class ObjectIsClosed
  The object is closed.
• class StrategyError
  A StrategyError object is thrown when the underlying implementation of this interface encounters an error.
• class NotImplemented
  A NotImplemented object is thrown when the underlying implementation of this interface has not or could not be created.
Functions

- string errorStr ()

D.1.1 Detailed Description

Exceptions, and other error handling. The Error package contains classes for exceptions, and functions used for error handling, including signals generated by a process.

D.1.2 Function Documentation

D.1.2.1 string BiometricEvaluation::Error::errorStr ()

Convert the value of errno to a human-readable error message.

Returns

The current error message specified by errno.

D.2 BiometricEvaluation::Finger Namespace Reference

Biometric information relating to finger images and derived information.

Classes

- class PatternClassification
  Pattern classification codes.
- class Position
  Finger position codes.
- class Impression
  Finger and palm impression types.
- class FingerImageCode

Typedefs

- typedef std::vector<Position::Kind> PositionSet
- typedef std::map<Position::Kind, FingerImageCode::Kind> PositionDescriptors

Functions

- std::ostream & operator<< (std::ostream &, const Finger::PatternClassification::Kind &)
  Output stream overload for PatternClassification::Kind.
- std::ostream & operator<< (std::ostream &, const Position::Kind &)
- std::ostream & operator<< (std::ostream &, const Impression::Kind &)
- std::ostream & operator<< (std::ostream &, const FingerImageCode::Kind &)
D.2.1 Detailed Description

Biometric information relating to finger images and derived information. The Finger package gathers all finger related matters, including classes to represent finger minutiae and helper functions for conversion between biometric representations. Contained within this namespace are classes to represent specific record formats, such as ANSI/NIST finger image records.

D.3 BiometricEvaluation::Image Namespace Reference

Basic information relating to images.

Classes

- class CompressionAlgorithm
  *Image compression algorithms.*
- struct Coordinate
  *A structure to contain a two-dimensional coordinate without a specified origin.*
- struct Size
  *A structure to represent the size of an image, in pixels.*
- struct Resolution
  *A structure to represent the resolution of an image.*
- class Image
  *Represent attributes common to all images.*
- class Raw
  *An image with no encoding or compression.*

Typedefs

- typedef struct Coordinate Coordinate
- typedef std::vector<Image::Coordinate> CoordinateSet
- typedef struct Size Size
- typedef struct Resolution Resolution

Functions

- std::ostream & operator<< (std::ostream &, const CompressionAlgorithm::Kind &)
- std::ostream & operator<< (std::ostream &, const Coordinate &)
- std::ostream & operator<< (std::ostream &stream, const CoordinateSet &coordinates)
  *Output stream overload for CoordinateSet.*
- std::ostream & operator<< (std::ostream &, const Size &)
- std::ostream & operator<< (std::ostream &, const Resolution &)
- std::ostream & operator<< (std::ostream &stream, const Resolution::Kind &kind)
- float distance (const Coordinate &p1, const Coordinate &p2)
  *Calculate the distance between two points.*
D.3.1 Detailed Description

Basic information relating to images. Classes and methods for manipulating images.

The Image package gathers all image related matters, including classes to represent an image, coordinates, and functions for conversion between biometric representations.

D.3.2 Function Documentation

D.3.2.1 std::ostream& BiometricEvaluation::Image::operator<<( std::ostream & stream, const CoordinateSet & coordinates )

Output stream overload for CoordinateSet.

Parameters

| in | stream | Stream on which to append formatted CoordinateSet information. |
|    | coordinates | CoordinateSet information to append to stream. |

Returns

stream with a coordinates textual representation appended.

D.3.2.2 float BiometricEvaluation::Image::distance( const Coordinate & p1, const Coordinate & p2 )

Calculate the distance between two points.

Parameters

| in | p1 | First point. |
|    | p2 | Second point. |

Returns

Distance between p1 and p2.

D.4 BiometricEvaluation::IO Namespace Reference

Input/Output functionality.

Namespaces

- namespace Utility

Classes

- class FileRecordStore
- class Properties
  
  Maintain key/value pairs of strings, with each property matched to one value.
- class PropertiesFile
  
  A Properties object persisted in an file on disk.
- class RecordStore
  
  A class to represent a data storage mechanism.
D.5 BiometricEvaluation::IO::Utility Namespace Reference

Typedefs

• typedef map&lt; string, string &gt; PropertiesMap

Variables

• static const uint8_t READWRITE = 0
• static const uint8_t READONLY = 1

D.4.1 Detailed Description

Input/Output functionality. The IO package contains classes and functions used to abstract input and output operations and provide for robust error handling on behalf of the application.

D.4.2 Typedef Documentation

D.4.2.1 typedef map&lt; string, string &gt; BiometricEvaluation::IO::PropertiesMap

Internal structure used for storing property keys/values

D.4.3 Variable Documentation

D.4.3.1 const uint8_t BiometricEvaluation::IO::READWRITE = 0 [static]

Constant indicating the state of an object that manages some underlying file is accessible for reading and writing.

D.4.3.2 const uint8_t BiometricEvaluation::IO::READONLY = 1 [static]

Constant indicating the state of an object that manages some underlying file is accessible for reading only.

D.5 BiometricEvaluation::IO::Utility Namespace Reference

Functions

• void removeDirectory (const string &directory, const string &prefix) throw (Error::ObjectDoesNotExist, Error::StrategyError)
  Remove a directory.
• void setAsideName (const string &name) throw (Error::ObjectDoesNotExist, Error::StrategyError)
  Set aside a file or directory name.
• uint64_t getFileSize (const string &pathname) throw (Error::ObjectDoesNotExist, Error::StrategyError)
• bool fileExists (const string &pathname) throw (Error::StrategyError)
• bool pathIsDirectory (const string &pathname) throw (Error::StrategyError)
• bool validateRootName (const string &name)
• bool constructAndCheckPath (const string &name, const string &parentDir, string &fullPath)
• int makePath (const string &path, const mode_t mode)
  Create an entire directory tree.
• Memory::uint8Array readFile (const string &path, ios_base::openmode mode=ios_base::binary) throw (Error::ObjectDoesNotExist, Error::StrategyError)
Read the contents of a file into a buffer.
- `void writeFile (const uint8_t *data, const size_t size, const string &path, ios_base::openmode mode=ios_base::binary)` throw (Error::ObjectExists, Error::StrategyError)
  - Write the contents of a buffer to a file.
- `void writeFile (const Memory::uint8Array data, const string &path, ios_base::openmode mode=ios_base::binary)` throw (Error::ObjectExists, Error::StrategyError)
  - Write the contents of a buffer to a file.
- `bool isReadable (const string &pathname)`
  - Determine if a file can be opened with read permission.
- `bool isWritable (const string &pathname)`
  - Determine if a file can be opened with read/write permission.
- `string createTemporaryFile (const string &prefix="", const string &parentDir="/tmp` throw (Error::FileError, Error::MemoryError)
  - Create a temporary file.
- `FILE * createTemporaryFile (string &path, const string &prefix="", const string &parentDir="/tmp") throw (Error::FileError, Error::MemoryError)
  - Create a temporary file.

D.5.1 Detailed Description

A class containing utility functions used for IO operations. These functions are class methods.

D.5.2 Function Documentation

D.5.2.1 `void BiometricEvaluation::IO::Utility::removeDirectory (const string &directory, const string &prefix)` throw (Error::ObjectDoesNotExist, Error::StrategyError)

Remove a directory.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>directory</th>
<th>The name of the directory to be removed, without a preceding path.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>prefix</td>
<td>The path leading to the directory.</td>
</tr>
</tbody>
</table>

**Exceptions**

<table>
<thead>
<tr>
<th>Error::ObjectDoesNotExist</th>
<th>The named directory does not exist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system, or the directory name or prefix is malformed.</td>
</tr>
</tbody>
</table>

D.5.2.2 `void BiometricEvaluation::IO::Utility::setAsideName (const string &name)` throw (Error::ObjectDoesNotExist, Error::StrategyError)

Set aside a file or directory name.

A file or directory is renamed in a sequential manner. For example, if directory foo is set aside, it will be renamed foo.1. If foo is recreated by the application, and again set aside, it will be renamed foo.2. There is a limit of uint16_t max attempts at creating a set aside name.

**Parameters**

Generated on July 3, 2012
The path name of the file or directory to be set aside.

Exceptions

<table>
<thead>
<tr>
<th>Error::ObjectDoesNotExist</th>
<th>The named object does not exist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system, the name or prefix is malformed, or the maximum number of attempts was reached.</td>
</tr>
</tbody>
</table>

Get the size of a file.

Parameters

| in | pathname | The name of the file to be sized; can be a complete path. |

Returns

The file size.

Exceptions

<table>
<thead>
<tr>
<th>Error::ObjectDoesNotExist</th>
<th>The named directory does not exist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system, or pathname is malformed.</td>
</tr>
</tbody>
</table>

Indicate whether a file exists.

Parameters

| in | pathname | The name of the file to be checked; can be a complete path. |

Returns

true if the file exists, false otherwise.

Exceptions

| Error::StrategyError | An error occurred when using the underlying storage system, or pathname is malformed. |

Check whether or not a string is valid as a name for a rooted entity, such as a RecordStore or other type of container that is persistent within the file system. Notably, name cannot contain path name separators ('/' and '\') or begin with whitespace.

Parameters
D.5 BiometricEvaluation::IO::Utility Namespace Reference

| in | name | The proposed name for the entity. |
|----------------|

**Returns**

true if the name is acceptable, false otherwise.

D.5.2.6 bool BiometricEvaluation::IO::Utility::constructAndCheckPath ( const string & name, const string & parentDir, string & fullPath )

Construct a full path for a rooted entity, and return true if that path exists; false otherwise.

**Parameters**

| in | name | The proposed name for the entity; cannot be a pathname. |
| in | parentDir | The name of the directory to contain the entity. |
| out | fullPath | The complete path to the new entity, when when true is returned; ambiguous when false is returned. |

**Returns**

true if the named entity is present in the file system, false otherwise.

D.5.2.7 int BiometricEvaluation::IO::Utility::makePath ( const string & path, const mode_t mode )

Create an entire directory tree.

All intermediate nodes are created if they don’t exist.

**Parameters**

| in | path | The path to create. |
| in | mode | The permission mode of each element in the path. See chmod(2). |

**Returns**

0 on success, non-zero otherwise, and errno can be checked.

D.5.2.8 Memory::uint8Array BiometricEvaluation::IO::Utility::readFile ( const string & path, ios_base::openmode mode = ios_base::binary ) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Read the contents of a file into a buffer.

**Parameters**

| path | Path to a file to be read. |
| mode | Bitwise OR’d arguments to send to the file stream constructor. |

**Returns**

Contents of path in a buffer.

**Exceptions**

| Error::ObjectDoesNotExist | path does not exist. |
| Error::StrategyError | An error occurred when using the underlying storage system. |
D.5.2.9  void BiometricEvaluation::IO::Utility::writeFile ( const uint8_t * data, const size_t size, const string & path, 
            ios_base::openmode mode = ios_base::binary ) throw (Error::ObjectExists, Error::StrategyError)

Write the contents of a buffer to a file.

Parameters

<table>
<thead>
<tr>
<th>data</th>
<th>Data buffer to write.</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Size of data.</td>
</tr>
<tr>
<td>path</td>
<td>Path to file to create with contents of data.</td>
</tr>
<tr>
<td>mode</td>
<td>Bitwise OR’d arguments to send to the file stream constructor.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>ObjectExists</th>
<th>path exists but truncate not set, or path exists and is a directory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrategyError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

D.5.2.10  void BiometricEvaluation::IO::Utility::writeFile ( const Memory::uint8Array data, const string & path,  
            ios_base::openmode mode = ios_base::binary ) throw (Error::ObjectExists, Error::StrategyError)

Write the contents of a buffer to a file.

Parameters

<table>
<thead>
<tr>
<th>data</th>
<th>Data buffer to write.</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Path to file to create with contents of data.</td>
</tr>
<tr>
<td>mode</td>
<td>Bitwise OR’d arguments to send to the file stream constructor.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>ObjectExists</th>
<th>path exists but truncate not set, or path exists and is a directory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrategyError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

D.5.2.11  bool BiometricEvaluation::IO::Utility::isReadable ( const string & pathname )

Determine if a file can be opened with read permission.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>pathname</th>
<th>Path to the file to check.</th>
</tr>
</thead>
</table>

Returns
ture if the file can be opened with read permission, false otherwise.

Note

Could return true if the file does not exist, though fileExists() will return false if you do not have read permission.

See also

BiometricEvaluation::IO::Utility::fileExists()
D.5.2.12 bool BiometricEvaluation::IO::Utility::isWritable ( const string & pathname )

Determine if a file can be opened with read/write permission.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>pathname</th>
<th>Path to the file to check.</th>
</tr>
</thead>
</table>

**Returns**

true if the file can be opened with write permission, false otherwise.

**Note**

Could return true if the file does not exist, though fileExists() will return false if you do not have read permission.

**See also**

BiometricEvaluation::IO::Utility::fileExists()

---

D.5.2.13 string BiometricEvaluation::IO::Utility::createTemporaryFile ( const string & prefix = "", const string & parentDir = "/tmp" ) throw (Error::FileError, Error::MemoryError)

Create a temporary file.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>prefix</th>
<th>String to be prefixed to the random temporary name.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parentDir</td>
<td>Where to place the temporary file.</td>
</tr>
</tbody>
</table>

**Exceptions**

| Error::FileError | Could not create or close temporary file. |
| Error::MemoryError | Error allocating memory for file name. |

**Returns**

Path to temporary file.

**Note**

Exclusivity is not guaranteed for the path returned, since the exclusive descriptor is closed before returning.

---

D.5.2.14 FILE* BiometricEvaluation::IO::Utility::createTemporaryFile ( string & path, const string & prefix = "", const string & parentDir = "/tmp" ) throw (Error::FileError, Error::Memory::MemoryError)

Create a temporary file.

Exclusivity to the file stream is guaranteed.

**Parameters**
D.6 BiometricEvaluation::Text Namespace Reference

<table>
<thead>
<tr>
<th>out</th>
<th>path</th>
<th>Reference to a string that will hold the path to the opened temporary file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>prefix</td>
<td>String to be prefixed to the random temporary name.</td>
</tr>
<tr>
<td>in</td>
<td>parentDir</td>
<td>Where to place the temporary file.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Error::FileError</th>
<th>Could not create or close temporary file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::MemoryError</td>
<td>Error allocating memory for file name.</td>
</tr>
</tbody>
</table>

Returns

Open file stream to path.

Note

Caller must fclose(3) the returned stream.

D.6 BiometricEvaluation::Text Namespace Reference

Text processing for string objects.

Functions

- void removeLeadingTrailingWhitespace (string &s)
  Remove lead and trailing white space from a string object.
- string dirname (const string &path)
  Extract the directory part of a pathname.

D.6.1 Detailed Description

Text processing for string objects. The Text package contains a set of functions for the processing of strings: removing leading and trailing whitespace, computing a digest, and other utility functions.

D.6.2 Function Documentation

D.6.2.1 string BiometricEvaluation::Text::dirname ( const string & path )

Extract the directory part of a pathname.

Parameters

| in  | path | Path from which to extract the directory portion. |

Returns

Directory portion of path.

D.7 FPVTE2012 Namespace Reference

Contains all the structures and functions used by the API.

Generated on July 3, 2012
Classes

- struct FingerImage
  
  Object containing the fingerprint image and corresponding attributes.

- struct FingerOutput
  
  Object for output data related to an extracted template.

- struct Candidate
  
  Object defining format of candidate reporting.

- class StatusCode

- struct ReturnStatus
  
  A structure to contain information about a failure by the software under test.

- class SDKInterface

Typedefs

- typedef struct FingerImage FingerImage

- typedef std::vector<FingerImage> FingerImageSet

- typedef struct FingerOutput FingerOutput

- typedef std::vector<FingerOutput> FingerOutputSet

- typedef struct Candidate Candidate

- typedef std::vector<Candidate> CandidateSet

- typedef struct ReturnStatus ReturnStatus

- typedef tr1::shared_ptr<SDKInterface> SDKIptr

Functions

- std::ostream & operator<< (std::ostream &s, const FPVTE2012::FingerOutput &fo)

- std::ostream & operator<< (std::ostream &s, const FPVTE2012::StatusCode::Kind &sc)

- std::ostream & operator<< (std::ostream &s, const FPVTE2012::ReturnStatus &rs)

D.7.1 Detailed Description

Contains all the structures and functions used by the API.

D.7.2 Typedef Documentation

D.7.2.1 typedef std::vector<FingerImage> FPVTE2012::FingerImageSet

Object representing a set of Finger records. Using this object anywhere from one to ten fingerprints will be input for a single test subject.

D.7.2.2 typedef std::vector<FingerOutput> FPVTE2012::FingerOutputSet

Object representing a set of Finger output records.

D.7.2.3 typedef std::vector<Candidate> FPVTE2012::CandidateSet

Object representing a list of Candidates. This object allows the search function to return a candidate list of 1:L as defined by the evaluation protocol.
D.7 FPVTE2012 Namespace Reference

D.7.3 Function Documentation

D.7.3.1 std::ostream& FPVTE2012::operator<< ( std::ostream & s, const FPVTE2012::FingerOutput & fo )

Output stream operator for a FingerOutput object.

D.7.3.2 std::ostream& FPVTE2012::operator<< ( std::ostream & s, const FPVTE2012::StatusCode::Kind & sc )

Output stream operator for a StatusCode object.

D.7.3.3 std::ostream& FPVTE2012::operator<< ( std::ostream & s, const FPVTE2012::ReturnStatus & rs )

Output stream operator for a ReturnStatus object.
Appendix E

Class Documentation

E.1 BiometricEvaluation::Memory::AutoArray< T > Class Template Reference

A C-style array wrapped in the facade of a C++ STL container.

#include <be_memory_autoarray.h>

Public Types

- typedef T value_type
- typedef size_t size_type
- typedef T * iterator
- typedef const T * const_iterator
- typedef T & reference
- typedef const T & const_reference

Public Member Functions

- operator T * ()
  Convert AutoArray to T array.
- operator const T * () const
  Convert AutoArray to const T array.
- reference operator[ ] (ptrdiff_t index)
  Subscripting operator overload with unchecked access.
- const_reference operator[ ] (ptrdiff_t index) const
  Const subscripting operator overload with unchecked access.
- reference at (ptrdiff_t index) throw (out_of_range)
  Subscribe into the AutoArray with checked access.
- const_reference at (ptrdiff_t index) const throw (out_of_range)
  Const subscribe into the AutoArray with checked access.
- iterator begin ()
  Obtain an iterator to the beginning of the AutoArray.
- const_iterator begin () const
  Obtain an iterator to the beginning of the AutoArray.
- iterator end ()
Obtain an iterator to the end of the `AutoArray`.

- `const_iterator end () const`
  Obtain an iterator to the end of the `AutoArray`.

- `size_type size () const`
  Obtain the number of accessible elements.

- void `resize (size_type new_size, bool free=false) throw (Error::MemoryError)`
  Change the number of accessible elements.

- void `copy (const_iterator buffer)`
  Deep-copy the contents of a buffer into this `AutoArray`.

- void `copy (const_iterator buffer, size_type size)`
  Deep-copy the contents of a buffer into this `AutoArray`.

- `AutoArray (size_type size=0) throw (Error::MemoryError)`
  Construct an `AutoArray`.

- `AutoArray (const AutoArray &copy) throw (Error::MemoryError)`
  Construct an `AutoArray`.

- `AutoArray & operator= (const AutoArray &other) throw (Error::MemoryError)`
  Assignment operator overload performing a deep copy.

- `~AutoArray ()`

### E.1.1 Detailed Description

template<class T> class BiometricEvaluation::Memory::AutoArray< T >

A C-style array wrapped in the facade of a C++ STL container.

### E.1.2 Member Typedef Documentation

#### E.1.2.1 template<class T> typedef T BiometricEvaluation::Memory::AutoArray< T >::value_type

Type of element

#### E.1.2.2 template<class T> typedef size_t BiometricEvaluation::Memory::AutoArray< T >::size_type

Type of subscripts, counts, etc.

#### E.1.2.3 template<class T> typedef T* BiometricEvaluation::Memory::AutoArray< T >::iterator

Iterator of element

#### E.1.2.4 template<class T> typedef const T* BiometricEvaluation::Memory::AutoArray< T >::const_iterator

Const iterator of element

#### E.1.2.5 template<class T> typedef T& BiometricEvaluation::Memory::AutoArray< T >::reference

Reference to element
E.1.2.6 template<class T> typedef const T& BiometricEvaluation::Memory::AutoArray<T>::const_reference

Const reference element

E.1.3 Constructor & Destructor Documentation

E.1.3.1 template<class T> BiometricEvaluation::Memory::AutoArray<T>::AutoArray ( size_type size = 0 ) throw (Error::MemoryError)

Construct an AutoArray.

Parameters

in size The number of elements this AutoArray should initially hold.

Exceptions

Error::MemoryError Could not allocate new memory.

E.1.3.2 template<class T> BiometricEvaluation::Memory::AutoArray<T>::AutoArray ( const AutoArray<T>& copy ) throw (Error::MemoryError)

Construct an AutoArray.

Parameters

in copy An AutoArray whose contents will be deep copied into the new AutoArray.

Exceptions

Error::MemoryError Could not allocate new memory.

E.1.3.3 template<class T> BiometricEvaluation::Memory::AutoArray<T>::~AutoArray ( )

Destructor

E.1.4 Member Function Documentation

E.1.4.1 template<class T> BiometricEvaluation::Memory::AutoArray<T>::operator T* ( )

Convert AutoArray to T array.

Returns
Pointer to the beginning of the underlying array storage.
E.1.4.2 template<class T> BiometricEvaluation::Memory::AutoArray<const T>::operator const T * ( ) const

Convert AutoArray to const T array.

**Returns**
Const pointer to the beginning of the underlying array storage.

E.1.4.3 template<class T> BiometricEvaluation::Memory::AutoArray<T>::reference BiometricEvaluation::Memory::AutoArray<T>::operator[ ] ( ptrdiff_t index )

Subscripting operator overload with unchecked access.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>index</th>
</tr>
</thead>
</table>
|    | Subscript into underlying storage.

**Returns**
Reference to the element at the specified index.

E.1.4.4 template<class T> BiometricEvaluation::Memory::AutoArray<const T>::const_reference BiometricEvaluation::Memory::AutoArray<const T>::operator[ ] ( ptrdiff_t index ) const

Const subscipting operator overload with unchecked access.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>index</th>
</tr>
</thead>
</table>
|    | Subscript into underlying storage.

**Returns**
Const reference to the element at the specified index.

E.1.4.5 template<class T> BiometricEvaluation::Memory::AutoArray<T>::reference BiometricEvaluation::Memory::AutoArray<T>::at ( ptrdiff_t index ) throw (out_of_range)

Subscript into the AutoArray with checked access.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>index</th>
</tr>
</thead>
</table>
|    | Subscript into underlying storage.

**Returns**
Reference to the element at the specified index.

**Exceptions**

<table>
<thead>
<tr>
<th>out_of_range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specified index is outside the bounds of this AutoArray.</td>
</tr>
</tbody>
</table>
E.1.4.6 template<class T> BiometricEvaluation::Memory::AutoArray<T>::const_reference
    BiometricEvaluation::Memory::AutoArray<T>::at ( ptrdiff_t index ) const throw (out_of_range)

Subscript into the AutoArray with checked access.

**Parameters**

<table>
<thead>
<tr>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscript into underlying storage.</td>
</tr>
</tbody>
</table>

**Returns**
Const reference to the element at the specified index.

**Exceptions**

<table>
<thead>
<tr>
<th>out_of_range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specified index is outside the bounds of this AutoArray.</td>
</tr>
</tbody>
</table>

E.1.4.7 template<class T> BiometricEvaluation::Memory::AutoArray<T>::iterator BiometricEvaluation::Memory::AutoArray<T>::begin ( )

Obtain an iterator to the beginning of the AutoArray.

**Returns**
Iterator positioned at the first element of the AutoArray.

E.1.4.8 template<class T> BiometricEvaluation::Memory::AutoArray<T>::const_iterator BiometricEvaluation::Memory::AutoArray<T>::begin ( ) const

Obtain an iterator to the beginning of the AutoArray.

**Returns**
Const iterator positioned at the first element of the AutoArray.

E.1.4.9 template<class T> BiometricEvaluation::Memory::AutoArray<T>::iterator BiometricEvaluation::Memory::AutoArray<T>::end ( )

Obtain an iterator to the end of the AutoArray.

**Returns**
Iterator positioned at the one-past-last element of the AutoArray.

E.1.4.10 template<class T> BiometricEvaluation::Memory::AutoArray<T>::const_iterator BiometricEvaluation::Memory::AutoArray<T>::end ( ) const

Obtain an iterator to the end of the AutoArray.
Returns
Iterator positioned at the one-past-last element of the AutoArray.

E.1.4.11 template<class T> BiometricEvaluation::Memory::AutoArray<T>::size_type BiometricEvaluation::Memory::AutoArray<T>::size() const
Obtain the number of accessible elements.

Returns
Number of accessible elements.

Note
If resize() has been called, the value returned from size() may be smaller than the actual allocated size of the underlying storage.

E.1.4.12 template<class T> void BiometricEvaluation::Memory::AutoArray<T>::resize(size_type new_size, bool free = false) throw (Error::MemoryError)
Change the number of accessible elements.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>new_size</td>
</tr>
<tr>
<td>in</td>
<td>free</td>
</tr>
</tbody>
</table>

Exceptions
Error::MemoryError Problem allocating memory.

E.1.4.13 template<class T> void BiometricEvaluation::Memory::AutoArray<T>::copy(const_iterator buffer)
Deep-copy the contents of a buffer into this AutoArray.

Parameters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>buffer</td>
</tr>
</tbody>
</table>

Warning
If buffer is smaller in size than the current size of the AutoArray, you MUST call copy(const_iterator, size_type). This method must only be used when buffer is larger than or equal to the size of the AutoArray.
E.1.4.14  template<class T> void BiometricEvaluation::Memory::AutoArray<T>::copy(const_iterator buffer, size_type size)

Deep-copy the contents of a buffer into this AutoArray.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>buffer</th>
<th>An allocated buffer whose contents will be deep-copied into this object.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>size</td>
<td>The number of bytes from buffer that will be deep-copied.</td>
</tr>
</tbody>
</table>

Warning

size must be less than or equal to the size of buffer.

E.1.4.15  template<class T> BiometricEvaluation::Memory::AutoArray<T> & BiometricEvaluation::Memory::AutoArray<T>::operator=(const AutoArray<T> & other) throw (Error::MemoryError)

Assignment operator overload performing a deep copy.

Parameters

|   | other | AutoArray to be copied. |

Returns

Reference to a new AutoArray object, the lvalue AutoArray.

Exceptions

| Error::MemoryError | Could not allocate new memory. |

The documentation for this class was generated from the following file:

- be_memory_autoarray.h

E.2  FPVTE2012::Candidate Struct Reference

Object defining format of candidate reporting.

#include <fpvte2012.h>

Public Attributes

- string templateID
- double similarity
E.2.1 Detailed Description

Object defining format of candidate reporting.

Each candidate returned for a search subject will report the information defined in this structure. Candidates should be sorted into ascending order of match probability.

Parameters

<table>
<thead>
<tr>
<th>templateID</th>
<th>The Template ID of the candidate. The ID returned for the candidate must be the same ID that was passed into enrollment finalization.</th>
</tr>
</thead>
<tbody>
<tr>
<td>similarity</td>
<td>The similarity score that the candidate and search templates are from the same subject.</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:

- fpvte2012.h

E.3 BiometricEvaluation::Image::CompressionAlgorithm Class Reference

Image compression algorithms.

#include <be_image.h>

Public Types

- enum Kind { None = 0, Facsimile = 1, WSQ20 = 2, JPEGB = 3, JPEGL = 4, JP2 = 5, JP2L = 6, PNG = 7, NetPBM = 8 }

E.3.1 Detailed Description

Image compression algorithms.

The documentation for this class was generated from the following file:

- be_image.h

E.4 BiometricEvaluation::Error::ConversionError Class Reference

Error when converting one object into another, a property value from string to int, for example.

#include <be_error_exception.h>

Inheritance diagram for BiometricEvaluation::Error::ConversionError:

```
+------------------+
| BiometricEvaluation::Error::Exception |
|      +------------------+           |
|      | BiometricEvaluation::Error::ConversionError |
```

Public Member Functions

- ConversionError ()
- ConversionError (string info)
E.4.1 Detailed Description

Error when converting one object into another, a property value from string to int, for example.

E.4.2 Constructor & Destructor Documentation

E.4.2.1 BiometricEvaluation::Error::ConversionError::ConversionError()

Construct a ConversionError object with the default information string.

Returns
The ConversionError object.

E.4.2.2 BiometricEvaluation::Error::ConversionError::ConversionError(string info)

Construct a ConversionError object with an information string appended to the default information string.

Returns
The ConversionError object.

The documentation for this class was generated from the following file:

- be_error_exception.h

---

E.5 BiometricEvaluation::Image::Coordinate Struct Reference

A structure to contain a two-dimensional coordinate without a specified origin.

#include <be_image.h>

Public Member Functions

- Coordinate (const uint32_t x=0, const uint32_t y=0, const float xDistance=0, const float yDistance=0)
  
  Create a Coordinate struct.

Public Attributes

- uint32_t x
- uint32_t y
- float xDistance
- float yDistance

E.5.1 Detailed Description

A structure to contain a two-dimensional coordinate without a specified origin.
E.5.2 Constructor & Destructor Documentation

E.5.2.1 BiometricEvaluation::Image::Coordinate::Coordinate ( const uint32_t x = 0, const uint32_t y = 0, const float xDistance = 0, const float yDistance = 0 )

Create a Coordinate struct.

Parameters

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>X-coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
<td>Y-coordinate</td>
</tr>
<tr>
<td></td>
<td>xDistance</td>
<td>X-coordinate distance from origin</td>
</tr>
<tr>
<td></td>
<td>yDistance</td>
<td>Y-coordinate distance from origin</td>
</tr>
</tbody>
</table>

E.5.3 Member Data Documentation

E.5.3.1 uint32_t BiometricEvaluation::Image::Coordinate::x

X-coordinate

E.5.3.2 uint32_t BiometricEvaluation::Image::Coordinate::y

Y-coordinate

E.5.3.3 float BiometricEvaluation::Image::Coordinate::xDistance

X-coordinate distance from origin

E.5.3.4 float BiometricEvaluation::Image::Coordinate::yDistance

Y-coordinate distance from origin

The documentation for this struct was generated from the following file:

- be_image.h

E.6 BiometricEvaluation::Error::DataError Class Reference

Error when reading data from an external source.

#include <be_error_exception.h>

Inheritance diagram for BiometricEvaluation::Error::DataError:

```
BiometricEvaluation::Error::DataError
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>BiometricEvaluation::Error::Exception</td>
</tr>
</tbody>
</table>
```
Public Member Functions

- DataError()
- DataError(string info)

E.6.1 Detailed Description

Error when reading data from an external source.

Typically occurs when reading data from a standard record, ANST/NIST 2000, for example, and a required field is missing, or a field has invalid data.

E.6.2 Constructor & Destructor Documentation

E.6.2.1 BiometricEvaluation::Error::DataError::DataError()

Construct a DataError object with the default information string.

Returns
The DataError object.

E.6.2.2 BiometricEvaluation::Error::DataError::DataError(string info)

Construct a DataError object with an information string appended to the default information string.

Returns
The DataError object.

The documentation for this class was generated from the following file:

- be_error_exception.h

E.7 BiometricEvaluation::Error::Exception Class Reference

The parent class of all BiometricEvaluation exceptions.

#include <be_error_exception.h>

Inheritance diagram for BiometricEvaluation::Error::Exception:
E.7 BiometricEvaluation::Error::Exception Class Reference

Public Member Functions

- Exception ()
- Exception (string info)
- string getInfo ()

E.7.1 Detailed Description

The parent class of all BiometricEvaluation exceptions.

The classes derived from this class will have a default information string set indicating the type of exception. Any additional information string is appended to that string.

E.7.2 Constructor & Destructor Documentation

E.7.2.1 BiometricEvaluation::Error::Exception::Exception ()

Construct an Exception object without an information string.

Returns
The Exception object.
E.8 BiometricEvaluation::Error::FileError Class Reference

E.7.2 Construct an Exception object with an information string.

Parameters

| in | info | The information string associated with the exception. |

Returns
The Exception object.

E.7.3 Member Function Documentation

E.7.3.1 string BiometricEvaluation::Error::Exception::getInfo()

Obtain the information string associated with the exception.

Returns
The information string.

The documentation for this class was generated from the following file:

- be_error_exception.h

E.8 BiometricEvaluation::Error::FileError Class Reference

File error when opening, reading, writing, etc.

#include <be_error_exception.h>

Inheritance diagram for BiometricEvaluation::Error::FileError:

```
BiometricEvaluation::Error::Exception
```

Public Member Functions

- FileError()
- FileError(string info)

E.8.1 Detailed Description

File error when opening, reading, writing, etc.
E.9 BiometricEvaluation::IO::FileRecordStore Class Reference

E.8.2 Constructor & Destructor Documentation

E.8.2.1 BiometricEvaluation::Error::FileError::FileError ( )

Construct a FileError object with the default information string.

Returns
The FileError object.

E.8.2.2 BiometricEvaluation::Error::FileError::FileError ( string info )

Construct a FileError object with an information string appended to the default information string.

Returns
The FileError object.

The documentation for this class was generated from the following file:

• be_error_exception.h

E.9 BiometricEvaluation::IO::FileRecordStore Class Reference

#include <be_io_filerecstore.h>

Inheritance diagram for BiometricEvaluation::IO::FileRecordStore:

BiometricEvaluation::IO::RecordStore

BiometricEvaluation::IO::FileRecordStore

Public Member Functions

• FileRecordStore (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::-StrategyError)
• FileRecordStore (const string &name, const string &parentDir, uint8_t mode=IO::READWRITE) throw (Error::ObjectDoes-NotExist, Error::StrategyError)
• uint64_t getSpaceUsed () const throw (Error::StrategyError)

  Obtain real storage utilization.

• void insert (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
• void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
• uint64_t read (const string &key, const void *const data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
• virtual void replace (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::-StrategyError)
• virtual uint64_t length (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
• void flush (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
• `uint64_t sequence (string &key, void *const data=NULL, int cursor=BE_RECSTORE_SEQ_NEXT)` throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Sequence through a RecordStore, returning the key/data pairs.

• void `setCursorAtKey (string &key)` throw (Error::ObjectDoesNotExist, Error::StrategyError)

• void `changeName (const string &name)` throw (Error::ObjectExists, Error::StrategyError)

### Protected Member Functions

• `string canonicalName (const string &name) const`

### Detailed Description

Class to represent the record store data storage mechanism implemented as files for each record.

**Note**

For the methods that take a key parameter, Error::StrategyError will be thrown if the key string is not compliant. A FileRecordStore has the additional requirement that a key name may not contain path delimiter characters ('/' and '\'), or begin with whitespace.

### Constructor & Destructor Documentation

#### E.9.2.1 BiometricEvaluation::IO::FileRecordStore::FileRecordStore

Create a new FileRecordStore, read/write mode.

**Parameters**

- **in name**  The name of the store.
- **in description**  The store’s description.
- **in parentDir**  The directory where the store is to be created.

**Exceptions**

<table>
<thead>
<tr>
<th>Error::ObjectExists</th>
<th>The store already exists.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when accessing the underlying file system.</td>
</tr>
</tbody>
</table>

#### E.9.2.2 BiometricEvaluation::IO::FileRecordStore::FileRecordStore

Open an existing FileRecordStore.

**Parameters**

- **in name**  The name of the store.
- **in parentDir**  The directory where the store is to be created.
- **in mode**  Open mode, read-only or read-write.

**Exceptions**

<table>
<thead>
<tr>
<th>Error::ObjectDoesNotExist</th>
<th>The store does not exist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when accessing the underlying file system.</td>
</tr>
</tbody>
</table>
E.9 BiometricEvaluation::IO::FileRecordStore Class Reference

E.9.3 Member Function Documentation

E.9.3.1 uint64_t BiometricEvaluation::IO::FileRecordStore::getSpaceUsed ( ) const throw (Error::StrategyError) [virtual]

Obtain real storage utilization.
The amount of disk space used, for example. This is the actual space allocated by the underlying storage mechanism, in bytes.

Returns
The amount of backing storage used by the RecordStore.

Exceptions

| Error::StrategyError | An error occurred when using the underlying storage system. |

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.9.3.2 void BiometricEvaluation::IO::FileRecordStore::insert ( const string & key, const void * const data, const uint64_t size ) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Insert a record into the store.

Parameters

| in key | The key of the record to be inserted. |
| in data | The data for the record. |
| in size | The size, in bytes, of the record. |

Exceptions

| Error::ObjectExists | A record with the given key is already present. |
| Error::StrategyError | An error occurred when using the underlying storage system. |

Implements BiometricEvaluation::IO::RecordStore.

E.9.3.3 void BiometricEvaluation::IO::FileRecordStore::remove ( const string & key ) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Remove a record from the store.

Parameters

| in key | The key of the record to be removed. |

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError | An error occurred when using the underlying storage system. |

Implements BiometricEvaluation::IO::RecordStore.
E.9.3.4  

uint64_t BiometricEvaluation::IO::FileRecordStore::read ( const string & key, void *const data ) const throw ( Error::ObjectDoesNotExist, Error::StrategyError ) [virtual]

Read a complete record from a store. Applications are responsible for allocating storage for the record’s data.

Parameters

| in | key | The key of the record to be read. |
| in | data | Pointer to where the data is to be written. |

Returns

The size of the record.

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError | An error occurred when using the underlying storage system. |

Implements BiometricEvaluation::IO::RecordStore.

E.9.3.5  

virtual void BiometricEvaluation::IO::FileRecordStore::replace ( const string & key, const void *const data, const uint64_t size ) throw ( Error::ObjectDoesNotExist, Error::StrategyError ) [virtual]

Replace a complete record in a store.

Parameters

| in | key | The key of the record to be replaced. |
| in | data | The data for the record. |
| in | size | The size of data. |

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError | An error occurred when using the underlying storage system. |

Implements BiometricEvaluation::IO::RecordStore.

E.9.3.6  

virtual uint64_t BiometricEvaluation::IO::FileRecordStore::length ( const string & key ) const throw ( Error::ObjectDoesNotExist, Error::StrategyError ) [virtual]

Return the length of a record.

Parameters

| in | key | The key of the record. |

Returns

The record length.

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError | An error occurred when using the underlying storage system. |

Implements BiometricEvaluation::IO::RecordStore.
E.9.3.7 void BiometricEvaluation::IO::FileRecordStore::flush ( const string & key ) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Commit the record’s data to storage.

Parameters

- **in** | **key** | The key of the record to be flushed.

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError | An error occurred when using the underlying storage system. |

Implements BiometricEvaluation::IO::RecordStore.

E.9.3.8 uint64_t BiometricEvaluation::IO::FileRecordStore::sequence ( string & key, void *const data = NULL, int cursor = BE_RECSTORE_SEQ_NEXT ) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Sequence through a RecordStore, returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the sequencor to return the next record. The starting point is typically the first record, and is set to that when the RecordStore object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

- **out** | **key** | The key of the currently sequenced record. |
- **in** | **data** | Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted. |
- **in** | **cursor** | The location within the sequence of the key/data pair to return. |

Returns

The length of the record currently in sequence.

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError | An error occurred when using the underlying storage system. |

Implements BiometricEvaluation::IO::RecordStore.

E.9.3.9 void BiometricEvaluation::IO::FileRecordStore::setCursorAtKey ( string & key ) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Set the sequence cursor to an arbitrary position within the RecordStore, starting at key. Key will be the first record returned from the next call to sequence().

Parameters

- **in** | **key** | The key of the record which will be returned by the first subsequent call to sequence(). |

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError | An error occurred when using the underlying storage system. |

Implements BiometricEvaluation::IO::RecordStore.
E.9.3.10 void BiometricEvaluation::IO::FileRecordStore::changeName ( const string & name ) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Change the name of the RecordStore.

**Parameters**

| in  | name | The new name for the RecordStore. |

**Exceptions**

| Error::StrategyError | An error occurred when using the underlying storage system, or the name is malformed. |

Reimplemented from BiometricEvaluation::IO::RecordStore.

The documentation for this class was generated from the following file:

- be_io_filerecstore.h

---

E.10 FPVTE2012::FingerImage Struct Reference

Object containing the fingerprint image and corresponding attributes.

#include <fpvte2012.h>

**Public Attributes**

- Finger::Position::Kind fgp
- Finger::Impression::Kind imp
- tr1::shared_ptr<Image::Raw> raw_image
- uint8_t nfiq

E.10.1 Detailed Description

Object containing the fingerprint image and corresponding attributes.

The fingerprint image object used to pass the image and attributes to the template extraction process.

**Parameters**

<table>
<thead>
<tr>
<th>fgp</th>
<th>Finger position.</th>
</tr>
</thead>
<tbody>
<tr>
<td>imp</td>
<td>Impression type for the fingerprint.</td>
</tr>
<tr>
<td>raw_image</td>
<td>Input image data.</td>
</tr>
<tr>
<td>nfiq</td>
<td>Image nfiq value with range from 1 (best) ... 5 (worst).</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:

- fpvte2012.h

---

E.11 BiometricEvaluation::Finger::FingerImageCode Class Reference

#include <be_finger.h>
Public Types

- enum Kind { EJI = 0, RolledTip, FullFingerRolled, FullFingerPlainLeft, FullFingerPlainCenter, FullFingerPlainRight, ProximalSegment, DistalSegment, MedialSegment, NA }

E.11.1 Detailed Description

Joint and tip codes.

The documentation for this class was generated from the following file:

- be_finger.h

E.12 FPVTE2012::FingerOutput Struct Reference

Object for output data related to an extracted template.

#include <fpvte2012.h>

Public Attributes

- Finger::Position::Kind fgp
- uint32_t template_size
- uint8_t image_quality
- bool core_location_present
- Image::Coordinate core_location
- bool delta_location_present
- Image::Coordinate delta_location

E.12.1 Detailed Description

Object for output data related to an extracted template.

Information need by the NIST driver to store the extracted template into a RecordStore and potentially for use in results analysis.

Parameters

<table>
<thead>
<tr>
<th>fgp</th>
<th>Finger position.</th>
</tr>
</thead>
<tbody>
<tr>
<td>template_size</td>
<td>The actual size of the extracted template.</td>
</tr>
<tr>
<td>image_quality</td>
<td>Image quality computed by the template extraction application. Valid range is 0-100 with 0 being lowest quality.</td>
</tr>
<tr>
<td>core_location_present</td>
<td>Whether or not core_location has been set.</td>
</tr>
<tr>
<td>core_location</td>
<td>If available, the core location of the fingerprint. This is an x,y pixel coordinate location in the image.</td>
</tr>
<tr>
<td>delta_location_present</td>
<td>Whether or not delta_location has been set.</td>
</tr>
<tr>
<td>delta_location</td>
<td>If available, the delta location of the fingerprint. This is an x,y pixel coordinate location in the image.</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:

- fpvte2012.h
E.13 BiometricEvaluation::Image::Image Class Reference

Represent attributes common to all images.

#include <be_image_image.h>

Inheritance diagram for BiometricEvaluation::Image::Image:

```
BiometricEvaluation::Image::Image
   `-- BiometricEvaluation::Image::Raw
```

### Public Member Functions

- **Image (const uint8_t *data, const uint64_t size, const Size dimensions, const uint32_t depth, const Resolution resolution, const CompressionAlgorithm::Kind compression)** throw (Error::DataError, Error::StrategyError)
  
  **Parent constructor for all Image classes.**

- **Image (const uint8_t *data, const uint64_t size, const CompressionAlgorithm::Kind compression)** throw (Error::DataError, Error::StrategyError)
  
  **Parent constructor for all Image classes.**

- **CompressionAlgorithm::Kind getCompressionAlgorithm () const**

  **Accessor for the CompressionAlgorithm of the image.**

- **Resolution getResolution () const**

  **Accessor for the resolution of the image.**

- **Memory::AutoArray<uint8_t> getData () const**

  **Accessor for the image data. The data returned is likely encoded in a specialized format.**

- **virtual Memory::AutoArray<uint8_t> getRawData () const =0 throw (Error::DataError)**

  **Accessor for the raw image data. The data returned should not be compressed or encoded.**

- **virtual Memory::AutoArray<uint8_t> getRawGrayscaleData (uint8_t depth=8) const =0 throw (Error::DataError, Error::ParameterError)**

  **Accessor for decompressed data in grayscale.**

- **Size getDimensions () const**

  **Accessor for the dimensions of the image in pixels.**

- **uint32_t getDepth () const**

  **Accessor for the color depth of the image in bits.**

### Static Public Member Functions

- **static uint64_t valueInColorspace (uint64_t color, uint64_t maxColorValue, uint8_t depth)**

  **Calculate an equivalent color value for a color in an alternate colorspace.**

### Static Public Attributes

- **static const uint32_t bitsPerComponent = 8**
Protected Member Functions

- void setResolution (const Resolution resolution)
  Mutator for the resolution of the image.
- void setDimensions (const Size dimensions)
  Mutator for the dimensions of the image in pixels.
- void setDepth (const uint32_t depth)
  Mutator for the color depth of the image in bits.

Protected Attributes

- Memory::AutoArray < uint8_t > _raw_data

E.13.1 Detailed Description

Represent attributes common to all images.

Images are represented by their size, depth, and resolution on the X and Y axes. The image data can be of any format, raw, JPEG, etc. Implementations of this abstraction provide the getRawData() method to convert image data to 'raw' format.

Image resolution is in pixels per centimeter, and the coordinate system has the origin at the upper left of the image.

E.13.2 Constructor & Destructor Documentation

E.13.2.1 BiometricEvaluation::Image::Image ( const uint8_t* data, const uint64_t size, const Size dimensions, const uint32_t depth, const Resolution resolution, const CompressionAlgorithm::Kind compression ) throw (Error::DataError, Error::StrategyError)

Parent constructor for all Image classes.

Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in data</td>
<td>The image data.</td>
</tr>
<tr>
<td>in size</td>
<td>The size of the image data, in bytes.</td>
</tr>
<tr>
<td>in dimensions</td>
<td>The width and height of the image in pixels.</td>
</tr>
<tr>
<td>in depth</td>
<td>The image depth, in bits-per-pixel.</td>
</tr>
<tr>
<td>in resolution</td>
<td>The resolution of the image</td>
</tr>
<tr>
<td>in compression</td>
<td>The CompressionAlgorithm of data.</td>
</tr>
</tbody>
</table>

Exceptions

| Error::StrategyError | Error manipulating data. |
| Error::StrategyError | Error while creating Image. |

E.13.2.2 BiometricEvaluation::Image::Image ( const uint8_t* data, const uint64_t size, const CompressionAlgorithm::Kind compression ) throw (Error::DataError, Error::StrategyError)

Parent constructor for all Image classes.

Parameters
The image data.

- **size**: The size of the image data, in bytes.
- **compression**: The `CompressionAlgorithm` of data.

**Exceptions**

- `Error::DataError`: Error manipulating data.
- `Error::StrategyError`: Error while creating `Image`.

### E.13.3 Member Function Documentation

#### E.13.3.1 `CompressionAlgorithm::Kind` BiometricEvaluation::Image::Image::getCompressionAlgorithm( ) const

Accessor for the `CompressionAlgorithm` of the image.

**Returns**

Type of compression used on the data that will be returned from `getData()`.

#### E.13.3.2 `Resolution` BiometricEvaluation::Image::Image::getResolution( ) const

Accessor for the resolution of the image.

**Returns**

`Resolution` struct

#### E.13.3.3 `Memory::AutoArray<uint8_t>` BiometricEvaluation::Image::Image::getData( ) const

Accessor for the image data. The data returned is likely encoded in a specialized format.

**Returns**

`Image` data.

Reimplemented in `BiometricEvaluation::Image::Raw`.

#### E.13.3.4 `virtual Memory::AutoArray<uint8_t>` BiometricEvaluation::Image::Image::getRawData( ) const throw (`Error::DataError`)

[`pure virtual`]

Accessor for the raw image data. The data returned should not be compressed or encoded.

**Returns**

`Raw` image data.

**Exceptions**

- `Error::DataError`: Error decompressing image data.

Implemented in `BiometricEvaluation::Image::Raw`.
E.13.3.5 virtual Memory::AutoArray<uint8_t> BiometricEvaluation::Image::getRawGrayscaleData ( uint8_t depth = 8 ) const throw (Error::DataError, Error::ParameterError) [pure virtual]

Accessor for decompressed data in grayscale.

Parameters

| depth | The desired bit depth of the resulting raw image. This value may either be 8 or 1. |

Returns

Raw image buffer.

Exceptions

| Error::DataError | Error decompressing image data. |
| Error::ParameterError | Invalid value for depth. |

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implemented in BiometricEvaluation::Image::Raw.

E.13.3.6 Size BiometricEvaluation::Image::getDimensions ( ) const

Accessor for the dimensions of the image in pixels.

Returns

Coordinate object containing dimensions in pixels.

E.13.3.7 uint32_t BiometricEvaluation::Image::getDepth ( ) const

Accessor for the color depth of the image in bits.

Returns

The color depth of the image (bit).

E.13.3.8 static uint64_t BiometricEvaluation::Image::valueInColorspace ( uint64_t color, uint64_t maxColorValue, uint8_t depth ) [static]

Calculate an equivalent color value for a color in an alternate colorspace.

Parameters
<table>
<thead>
<tr>
<th>color</th>
<th>Value for color in original colorspace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxColorValue</td>
<td>Maximum value for colors in original colorspace.</td>
</tr>
<tr>
<td>depth</td>
<td>Desired bit-depth of the new colorspace.</td>
</tr>
</tbody>
</table>

Returns
A value equivalent to color in depth-bit space.

E.13.3.9  void BiometricEvaluation::Image::Image::setResolution ( const Resolution resolution ) [protected]
Mutator for the resolution of the image.
Parameters

| in | resolution | Resolution struct. |

E.13.3.10 void BiometricEvaluation::Image::Image::setDimensions ( const Size dimensions ) [protected]
Mutator for the dimensions of the image in pixels.
Parameters

| in | dimensions | Dimensions of image (pixel). |

E.13.3.11 void BiometricEvaluation::Image::Image::setDepth ( const uint32_t depth ) [protected]
Mutator for the color depth of the image in bits.
Parameters

| in | depth | The color depth of the image (bit). |

E.13.4  Member Data Documentation
E.13.4.1 const uint32_t BiometricEvaluation::Image::Image::bitsPerComponent = 8 [static]
Number of bits per color component
E.13.4.2 Memory::AutoArray<
uint8_t> BiometricEvaluation::Image::Image::_raw_data [mutable, protected]
Raw image data, populated on demand
The documentation for this class was generated from the following file:

- be_image_image.h

Generated on July 3, 2012
E.14 BiometricEvaluation::Finger::Impression Class Reference

Finger and palm impression types.

#include <be_finger.h>

Public Types

- enum Kind {
  LiveScanPlain = 0, LiveScanRolled, NonLiveScanPlain, NonLiveScanRolled, LatentImpression, LatentTracing, LatentPhoto, LatentLift, LiveScanVerticalSwipe, LiveScanPalm, NonLiveScanPalm, LatentPalmImpression, LatentPalmTracing, LatentPalmPhoto, LatentPalmLift, LiveScanOpticalContactPlain, LiveScanOpticalContactRolled, LiveScanNonOpticalContactPlain, LiveScanNonOpticalContactRolled, LiveScanOpticalContactlessPlain, LiveScanOpticalContactlessRolled, Liv

E.14.1 Detailed Description

Finger and palm impression types.
The documentation for this class was generated from the following file:

- be_finger.h

E.15 BiometricEvaluation::Error::MemoryError Class Reference

An error occurred when allocating an object.

#include <be_error_exception.h>

Inheritance diagram for BiometricEvaluation::Error::MemoryError:

```
BiometricEvaluation::Error::MemoryError
|----------------|
| BiometricEvaluation::Error::Exception
```

Public Member Functions

- MemoryError ()
- MemoryError (string info)

E.15.1 Detailed Description

An error occurred when allocating an object.

E.15.2 Constructor & Destructor Documentation

E.15.2.1 BiometricEvaluation::Error::MemoryError ( )

Construct a MemoryError object with the default information string.
Returns
The MemoryError object.

E.15.2.2  BiometricEvaluation::Error::MemoryError::MemoryError ( string info )
Construct a MemoryError object with an information string appended to the default information string.

Returns
The MemoryError object.

The documentation for this class was generated from the following file:

• be_error_exception.h

E.16  BiometricEvaluation::Error::NotImplemented Class Reference

A NotImplemented object is thrown when the underlying implementation of this interface has not or could not be created.
#include <be_error_exception.h>
Inheritance diagram for BiometricEvaluation::Error::NotImplemented:

```
BiometricEvaluation::Error::Exception

BiometricEvaluation::Error::NotImplemented
```

Public Member Functions

• NotImplemented ()
• NotImplemented (string info)

E.16.1  Detailed Description

A NotImplemented object is thrown when the underlying implementation of this interface has not or could not be created.

E.16.2  Constructor & Destructor Documentation

E.16.2.1  BiometricEvaluation::Error::NotImplemented::NotImplemented ( )
Construct a NotImplemented object with the default information string.

Returns
The NotImplemented object.
E.16.2.2 BiometricEvaluation::Error::NotImplemented::NotImplemented ( string \textit{info} )

Construct a \texttt{NotImplemented} object with an information string appended to the default information string.

\textbf{Returns}
The \texttt{NotImplemented} object.

The documentation for this class was generated from the following file:

- \texttt{be_error_exception.h}

E.17 BiometricEvaluation::Error::ObjectDoesNotExist Class Reference

The named object does not exist.

\texttt{#include &lt;be_error_exception.h&gt;}

Inheritance diagram for BiometricEvaluation::Error::ObjectDoesNotExist:

![Inheritance diagram]

\textbf{Public Member Functions}

- \texttt{ObjectDoesNotExist ()}
- \texttt{ObjectDoesNotExist (string info)}

E.17.1 Detailed Description

The named object does not exist.

E.17.2 Constructor & Destructor Documentation

E.17.2.1 BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( )

Construct a \texttt{ObjectDoesNotExist} object with the default information string.

\textbf{Returns}
The \texttt{ObjectDoesNotExist} object.

E.17.2.2 BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( string \textit{info} )

Construct a \texttt{ObjectDoesNotExist} object with an information string appended to the default information string.
Returns
The ObjectDoesNotExist object.

The documentation for this class was generated from the following file:

- be_error_exception.h

E.18 BiometricEvaluation::Error::ObjectExists Class Reference

The named object exists and will not be replaced.

```
#include <be_error_exception.h>
```

Inheritance diagram for BiometricEvaluation::Error::ObjectExists:

```
BiometricEvaluation::Error::ObjectExists
  `-- BiometricEvaluation::Error::Exception
```

Public Member Functions

- ObjectExists ()
- ObjectExists (string info)

E.18.1 Detailed Description

The named object exists and will not be replaced.

E.18.2 Constructor & Destructor Documentation

E.18.2.1 BiometricEvaluation::Error::ObjectExists::ObjectExists ()

Construct a ObjectExists object with the default information string.

Returns
The ObjectExists object.

E.18.2.2 BiometricEvaluation::Error::ObjectExists::ObjectExists ( string info )

Construct a ObjectExists object with an information string appended to the default information string.

Returns
The ObjectExists object.
The documentation for this class was generated from the following file:

- be_error_exception.h

## E.19 BiometricEvaluation::Error::ObjectIsClosed Class Reference

The object is closed.

```cpp
#include <be_error_exception.h>
```

Inheritance diagram for BiometricEvaluation::Error::ObjectIsClosed:

```
BiometricEvaluation::Error::ObjectIsClosed
```

### Public Member Functions

- `ObjectIsClosed ()`
- `ObjectIsClosed (string info)`

### E.19.1 Detailed Description

The object is closed.

### E.19.2 Constructor & Destructor Documentation

#### E.19.2.1 BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ( )

Construct a `ObjectIsClosed` object with the default information string.

**Returns**
The `ObjectIsClosed` object.

#### E.19.2.2 BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ( string info )

Construct a `ObjectIsClosed` object with an information string appended to the default information string.

**Returns**
The `ObjectIsClosed` object.

The documentation for this class was generated from the following file:

- be_error_exception.h
The object is already opened.

#include <be_error_exception.h>

Inheritance diagram for BiometricEvaluation::Error::ObjectIsOpen:

```
BiometricEvaluation::Error::ObjectIsOpen
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>BiometricEvaluation::Error::Exception</td>
</tr>
</tbody>
</table>
```

Public Member Functions

- `ObjectIsOpen ()`
- `ObjectIsOpen (string info)`

E.20.1 Detailed Description

The object is already opened.

E.20.2 Constructor & Destructor Documentation

E.20.2.1 BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ( )

Construct a `ObjectIsOpen` object with the default information string.

**Returns**

The `ObjectIsOpen` object.

E.20.2.2 BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ( string info )

Construct a `ObjectIsOpen` object with an information string appended to the default information string.

**Returns**

The `ObjectIsOpen` object.

The documentation for this class was generated from the following file:

- `be_error_exception.h`

E.21 BiometricEvaluation::Error::ParameterError Class Reference

An invalid parameter was passed to a constructor or method.

#include <be_error_exception.h>
E.22 BiometricEvaluation::Finger::PatternClassification Class Reference

Pattern classification codes.

#include <be_finger.h>

Public Types

• enum Kind { PlainArch = 0, TentedArch, RadialLoop, UlnarLoop, PlainWhorl, CentralPocketLoop, DoubleLoop, AccidentalWhorl, Whorl, RightSlantLoop, LeftSlantLoop, Scar, Amputation, Unknown }
E.22.1 Detailed Description

Pattern classification codes.
The documentation for this class was generated from the following file:

- be_finger.h

E.23 BiometricEvaluation::Finger::Position Class Reference

Finger position codes.
#include <be_finger.h>

Public Types

- enum Kind { Unknown = 0, RightThumb = 1, RightIndex = 2, RightMiddle = 3, RightRing = 4, RightLittle = 5, LeftThumb = 6, LeftIndex = 7, LeftMiddle = 8, LeftRing = 9, LeftLittle = 10, PlainRightThumb = 11, PlainLeftThumb = 12, PlainRightFourFingers = 13, PlainLeftFourFingers = 14, LeftRightThumbs = 15, EJI = 19 }

E.23.1 Detailed Description

Finger position codes.
These codes match those in ANSI/NIST. Other minutiae formats may have to map codes into this set.
The documentation for this class was generated from the following file:

- be_finger.h

E.24 BiometricEvaluation::IO::Properties Class Reference

Maintain key/value pairs of strings, with each property matched to one value.
#include <be_io_properties.h>

Inheritance diagram for BiometricEvaluation::IO::Properties:

```
BiometricEvaluation::IO::Properties
    BiometricEvaluation::IO::PropertiesFile
```

Public Types

- typedef PropertiesMap::const_iterator Properties_iter
Public Member Functions

- `Properties (uint8_t mode=IO::READWRITE)`
  
  Construct a new Properties object.

- `Properties (const uint8_t ∗buffer, const size_t size, uint8_t mode=IO::READWRITE) throw (Error::StrategyError)`
  
  Construct a new Properties object from the contents of a buffer.

- `virtual void setProperty (const string &property, const string &value) throw (Error::StrategyError)`
  
  Set a property with a value.

- `virtual void setPropertyFromInteger (const string &property, int64_t value) throw (Error::StrategyError)`
  
  Set a property with an integer value.

- `virtual void setPropertyFromDouble (const string &property, double value) throw (Error::StrategyError)`
  
  Set a property with a double value.

- `virtual void removeProperty (const string &property) throw (Error::ObjectDoesNotExist, Error::StrategyError)`
  
  Remove a property.

- `virtual string getProperty (const string &property) const throw (Error::ObjectDoesNotExist)`
  
  Retrieve a property value as a string object.

- `virtual int64_t getPropertyAsInteger (const string &property) const throw (Error::ObjectDoesNotExist, Error::ConversionError)`
  
  Retrieve a property value as an integer value.

- `virtual double getPropertyAsDouble (const string &property) const throw (Error::ObjectDoesNotExist)`
  
  Retrieve a property value as a double value.

- `virtual ~Properties ()`

Protected Member Functions

- `uint8_t getMode () const`
  
  Obtain the mode of the Properties object.

- `void initWithBuffer (const Memory::uint8Array &buffer) throw (Error::StrategyError)`
  
  Initialize the PropertiesMap with the contents of a properly formatted buffer.

- `void initWithBuffer (const uint8_t ∗const buffer, size_t size) throw (Error::StrategyError)`
  
  Initialize the PropertiesMap with the contents of a properly formatted buffer.

- `Properties_iter begin () const`
  
  Obtain iterator to the first property.

- `Properties_iter end () const`
  
  Obtain iterator to one past the last property.

E.24.1 Detailed Description

Maintain key/value pairs of strings, with each property matched to one value.

E.24.2 Member Typedef Documentation

E.24.2.1 `typedef PropertiesMap::const_iterator BiometricEvaluation::IO::Properties::Properties_iter`

Convenience const iterator over a Properties
E.24 BiometricEvaluation::IO::Properties Class Reference

E.24.3 Constructor & Destructor Documentation

E.24.3.1 BiometricEvaluation::IO::Properties::Properties ( uint8_t mode = IO::READWRITE )

Construct a new Properties object.

Parameters

| in  | mode | The read/write mode of the object. |

Returns

An object representing the properties set.

E.24.3.2 BiometricEvaluation::IO::Properties::Properties ( const uint8_t * buffer, const size_t size, uint8_t mode = IO::READWRITE ) throw (Error::StrategyError)

Construct a new Properties object from the contents of a buffer.

The format of the buffer can be seen in PropertiesFile.

Parameters

| in  | buffer | A buffer that contains the contents of a Property file. |
| in  | size   | The size of buffer. |
| in  | mode   | The read/write mode of the object. |

Returns

An object representing the properties set.

Exceptions

Error::StrategyError  A line in the properties file is malformed.

E.24.3.3 virtual BiometricEvaluation::IO::Properties::~Properties ( ) [virtual]

Destructor

E.24.4 Member Function Documentation

E.24.4.1 virtual void BiometricEvaluation::IO::Properties::setProperty ( const string & property, const string & value ) throw (Error::StrategyError) [virtual]

Set a property with a value.

Both the property and value will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise, the property will be created.

Parameters

| in  | property | The name of the property to set. |
| in  | value    | The value associated with the property. |
BiometricEvaluation::IO::Properties Class Reference

Exceptions

| Error::StrategyError | The Properties object is read-only. |

E.24.4.2 virtual void BiometricEvaluation::IO::Properties::setPropertyFromInteger ( const string & property, int64_t value )
    throw (Error::StrategyError) [virtual]

Set a property with an integer value.
The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with
the new value; otherwise the property will be created.

Parameters

| in property | The name of the property to set. |
| in value    | The value associated with the property. |

Exceptions

| Error::StrategyError | The Properties object is read-only. |

E.24.4.3 virtual void BiometricEvaluation::IO::Properties::setPropertyFromDouble ( const string & property, double value )
    throw (Error::StrategyError) [virtual]

Set a property with a double value.
The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with
the new value; otherwise the property will be created.

Parameters

| in property | The name of the property to set. |
| in value    | The value associated with the property. |

Exceptions

| Error::StrategyError | The Properties object is read-only. |

E.24.4.4 virtual void BiometricEvaluation::IO::Properties::removeProperty ( const string & property )
    throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Remove a property.

Parameters

| in property | The name of the property to set. |

Exceptions

| Error::ObjectDoesNotExist | The named property does not exist. |
| Error::StrategyError      | The Properties object is read-only. |
Retrieve a property value as a string object.

**Parameters**

| in | property | The name of the property to get. |

**Exceptions**

| Error::ObjectDoesNotExist | The named property does not exist. |

Retrieve a property value as an integer value.

Integer value strings for properties can represent either decimal or hexadecimal values, which must be preceded with either "0x" or "0X".

**Parameters**

| in | property | The name of the property to get. |

**Exceptions**

| Error::ObjectDoesNotExist | The named property does not exist. |
| Error::ConversionError | The property value cannot be converted, usually due to non-numeric characters in the string. |

Retrieve a property value as a double value.

**Parameters**

| in | property | The name of the property to get. |

**Exceptions**

| Error::ObjectDoesNotExist | The named property does not exist. |

Obtain the mode of the Properties object.

**Returns**

Mode (IO::READONLY or IO::READWRITE)
Initialize the PropertiesMap with the contents of a properly formatted buffer.
This method ensures that the PropertiesMap contains only the properties found within the buffer.

**Parameters**

| buffer | Contents of a properties file. |

**Exceptions**

| Error::StrategyError | A line of the buffer is malformed. |

Initialize the PropertiesMap with the contents of a properly formatted buffer.
This method ensures that the PropertiesMap contains only the properties found within the buffer.

**Parameters**

| buffer | Contents of a properties file. |

| size | Size of the buffer. |

**Exceptions**

| Error::StrategyError | A line of the buffer is malformed. |

Obtain iterator to the first property.

**Returns**

Iterator to first property.

Obtain iterator to one past the last property.

**Returns**

Iterator one past the last property.

The documentation for this class was generated from the following file:

- be_io_properties.h

A Properties object persisted in an file on disk.

#include <be_io_propertiesfile.h>

Inheritance diagram for BiometricEvaluation::IO::PropertiesFile:

- BiometricEvaluation::IO::PropertiesFile
  - BiometricEvaluation::IO::Properties

Public Member Functions

- **PropertiesFile** (const string &filename, uint8_t mode=READWRITE) throw (Error::FileError, Error::StrategyError)
  
  Construct a new Properties object from an existing or to be created properties file. The constructor will create the file when it does not exist.

- **void sync** () throw (Error::FileError, Error::StrategyError)
  
  Write the properties to the underlying file, synchronizing the in-memory and on-disk versions.

- **void changeName** (const string &filename) throw (Error::StrategyError)
  
  Change the name of the Properties, which means changing the name of the underlying file that stores the properties. The empty string ("") can be used to indicate no backing file.

- **~PropertiesFile** ()

E.25.1 Detailed Description

A Properties object persisted in an file on disk.

An example file might look like this:

```
* Name = John Smith
* Age = 32
* Favorite Hex Number = 0xffff
*
```

For property keys and values, leading and trailing whitespace is removed, therefore the call

```cpp
props->setProperty(" My property ", " A Value ");
```

results in an entry in the property file as

```
* My property = A value
*
```

Therefore, the property names "Foo", " Foo", "Foo " are equivalent.

E.25.2 Constructor & Destructor Documentation

E.25.2.1 BiometricEvaluation::IO::PropertiesFile::PropertiesFile ( const string & filename, uint8_t mode = READWRITE ) throw (Error::FileError, Error::StrategyError)

Construct a new Properties object from an existing or to be created properties file. The constructor will create the file when it does not exist.
Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>The name of the file to store the properties.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mode</td>
<td>The read/write mode of the object.</td>
</tr>
</tbody>
</table>

Returns

An object representing the properties set.

Exceptions

<table>
<thead>
<tr>
<th>Error::StrategyError</th>
<th>A line in the properties file is malformed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::FileError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

E.25.2.2 BiometricEvaluation::IO::PropertiesFile::~PropertiesFile()

Destructor

E.25.3 Member Function Documentation

E.25.3.1 void BiometricEvaluation::IO::PropertiesFile::sync() throw (Error::FileError, Error::StrategyError)

Write the properties to the underlying file, synchronizing the in-memory and on-disk versions.

Exceptions

<table>
<thead>
<tr>
<th>Error::FileError</th>
<th>An error occurred when using the underlying storage system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>The object was constructed with NULL as the file name, or is read-only.</td>
</tr>
</tbody>
</table>

E.25.3.2 void BiometricEvaluation::IO::PropertiesFile::changeName(const string & filename) throw (Error::StrategyError)

Change the name of the Properties, which means changing the name of the underlying file that stores the properties. The empty string ("") can be used to indicate no backing file.

Note

No check is made that the file is writeable at this time.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>filename</th>
<th>The name of the properties file.</th>
</tr>
</thead>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Error::StrategyError</th>
<th>The object is read-only.</th>
</tr>
</thead>
</table>

The documentation for this class was generated from the following file:

- be_io_propertiesfile.h
E.26 BiometricEvaluation::Image::Raw Class Reference

An image with no encoding or compression.

#include <be_image_raw.h>

Inheritance diagram for BiometricEvaluation::Image::Raw:

```
BiometricEvaluation::Image::Image

BiometricEvaluation::Image::Raw
```

Public Member Functions

- Raw (const uint8_t *data, const uint64_t size, const Size dimensions, const unsigned int depth, const Resolution resolution)
- Memory::AutoArray< uint8_t > getData () const
  
  Accessor for the image data. The data returned is likely encoded in a specialized format.
- Memory::AutoArray< uint8_t > getRawData () const throw (Error::DataError)
  
  Accessor for the raw image data. The data returned should not be compressed or encoded.
- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)
  
  Accessor for decompressed data in grayscale.

E.26.1 Detailed Description

An image with no encoding or compression.

E.26.2 Member Function Documentation

E.26.2.1 Memory::AutoArray< uint8_t > BiometricEvaluation::Image::Raw::getData () const

Accessor for the image data. The data returned is likely encoded in a specialized format.

Returns

Image data.

Reimplemented from BiometricEvaluation::Image::Image.

E.26.2.2 Memory::AutoArray< uint8_t > BiometricEvaluation::Image::Raw::getRawData () const throw (Error::DataError)

[virtual]

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

Raw image data.
Exceptions

<table>
<thead>
<tr>
<th>Error::DataError</th>
<th>Error decompressing image data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::ParameterError</td>
<td>Invalid value for depth.</td>
</tr>
</tbody>
</table>

Implements BiometricEvaluation::Image::Image.

E.26.2.3 Memory::AutoArray<
uint8_t>
BiometricEvaluation::Image::Raw::
getRawGrayscaleData (uint8_t depth = 8)
const throw (Error::DataError, Error::ParameterError) [virtual]

Accessor for decompressed data in grayscale.

Parameters

| depth | The desired bit depth of the resulting raw image. This value may either be 8 or 1. |

Returns

Raw image buffer.

Exceptions

<table>
<thead>
<tr>
<th>Error::DataError</th>
<th>Error decompressing image data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::ParameterError</td>
<td>Invalid value for depth.</td>
</tr>
</tbody>
</table>

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.
The documentation for this class was generated from the following file:

- be_image_raw.h

E.27 BiometricEvaluation::IO::RecordStore Class Reference

A class to represent a data storage mechanism.

#include <be_io_recordstore.h>

Inheritance diagram for BiometricEvaluation::IO::RecordStore:

BiometricEvaluation::IO::RecordStore

BiometricEvaluation::IO::FileRecordStore

Generated on July 3, 2012
Public Member Functions

- **RecordStore** (const string &name, const string &description, const string &type, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- **RecordStore** (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- string getName () const
- string getDescription () const
- unsigned int getCount () const
- virtual void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- virtual void changeDescription (const string &description) throw (Error::StrategyError)
- virtual uint64_t getSpaceUsed () const throw (Error::StrategyError)
  
  Obtain real storage utilization.
- virtual void sync () const throw (Error::StrategyError)
- virtual void insert (const string &key, const void *const data, const uint64_t size)=0 throw (Error::ObjectExists, Error::StrategyError)
- virtual void insert (const string &key, const Memory::uint8Array &data) throw (Error::ObjectExists, Error::StrategyError)
- virtual void remove (const string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t read (const string &key, void *const data) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t read (const string &key, Memory::uint8Array &data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
  
  Read a complete record from a store.
- virtual void replace (const string &key, const void *const data, const uint64_t size)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void replace (const string &key, const Memory::uint8Array &data) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t length (const string &key) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void flush (const string &key) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t sequence (string &key, void *const data=NULL, int cursor=BE_RECSTORE_SEQ_NEXT)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
  
  Sequence through a RecordStore, returning the key/data pairs.
- virtual uint64_t sequence (string &key, Memory::uint8Array &data, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)
  
  Sequence through a RecordStore, returning the key/data pairs.
- virtual void setCursorAtKey (string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

Static Public Member Functions

- static tr1::shared_ptr< RecordStore > openRecordStore (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
  
  Open an existing RecordStore and return a managed pointer to the the object representing that store.
- static tr1::shared_ptr< RecordStore > createRecordStore (const string &name, const string &description, const string &type, const string &destDir) throw (Error::ObjectExists, Error::StrategyError)
  
  Create a new RecordStore and return a managed pointer to the the object representing that store.
- static void removeRecordStore (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- static void mergeRecordStores (const string &mergedName, const string &mergedDescription, const string &parentDir, const string &type, RecordStore *recordStores[], size_t numRecordStores) throw (Error::ObjectExists, Error::StrategyError)
- static void mergeRecordStores (const string &mergedName, const string &mergedDescription, const string &parentDir, const string &type, tr1::shared_ptr< RecordStore > recordStores[], size_t numRecordStores) throw (Error::ObjectExists, Error::StrategyError)
### Static Public Attributes

- static const string INVALIDKEYCHARS
- static const char KEY_SEGMENT_SEPARATOR = '
'- static const uint64_t KEY_SEGMENT_START = 1
- static const string CONTROLFILENAME
- static const string NAMEPROPERTY
- static const string DESCRIPTIONPROPERTY
- static const string COUNTPROPERTY
- static const string TYPEPROPERTY
- static const string BERKELEYDBTYPE
- static const string ARCHIVETYPE
- static const string FILETYPE
- static const string SQLITETYPE
- static const int BE_RECSTORE_SEQ_START = 1
- static const int BE_RECSTORE_SEQ_NEXT = 2

### Protected Member Functions

- uint8_t getMode() const
- string getDirectory() const
- string getParentDirectory() const
- string canonicalName(const string &name) const
- int getCursor() const
- void setCursor(int cursor)
- bool validateKeyString(const string &key) const

### Static Protected Member Functions

- static string genKeySegName(const string &key, const uint64_t segnum)

  Generate key segment names.

### E.27.1 Detailed Description

A class to represent a data storage mechanism.

A RecordStore is an abstraction that associates keys with a specific record. Implementations of this abstraction can store the records in any format supported by the operating system, such as files or databases, rooted in the file system.

Certain characters are prohibited in the key string. See IO::RecordStore::INVALIDKEYCHARS. A key string cannot begin with the space character.

See also

- IO::ArchiveRecordStore, IO::DBRecordStore, IO::FileRecordStore.
E.27.2 Constructor & Destructor Documentation

E.27.2.1 BiometricEvaluation::IO::RecordStore::RecordStore ( const string & name, const string & description, const string & type, const string & parentDir ) throw (Error::ObjectExists, Error::StrategyError)

Constructor to create a new RecordStore.

Parameters

| in   | name   | The name of the RecordStore to be created. |
| in   | description | The text used to describe the store. |
| in   | type    | The type of RecordStore. |
| in   | parentDir | Where, in the file system, the store is to be rooted. This directory must exist. |

Returns

An object representing the new, empty store.

Exceptions

- **Error::ObjectExists** The store was previously created, or the directory where it would be created exists.
- **Error::StrategyError** An error occurred when using the underlying storage system, or the name is malformed.

E.27.2.2 BiometricEvaluation::IO::RecordStore::RecordStore ( const string & name, const string & parentDir, uint8_t mode = READWRITE ) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Constructor to open an existing RecordStore.

Parameters

| in   | name   | The name of the store to be opened. |
| in   | parentDir | Where, in the file system, the store is rooted. |
| in   | mode    | The type of access a client of this RecordStore has. |

Returns

An object representing the existing store.

Exceptions

- **Error::ObjectDoesNotExist** The RecordStore does not exist.
- **Error::StrategyError** An error occurred when using the underlying storage system, or the name is malformed.

E.27.3 Member Function Documentation

E.27.3.1 string BiometricEvaluation::IO::RecordStore::getName ( ) const

Return the name of the RecordStore.

Returns

The RecordStore’s name.
E.27.3.2  string BiometricEvaluation::IO::RecordStore::getDescription ( ) const

Obtain a textual description of the RecordStore.

**Returns**
The RecordStore’s description.

E.27.3.3  unsigned int BiometricEvaluation::IO::RecordStore::getCount ( ) const

Obtain the number of items in the RecordStore.

**Returns**
The number of items in the RecordStore.

E.27.3.4  virtual void BiometricEvaluation::IO::RecordStore::changeName ( const string & name ) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Change the name of the RecordStore.

**Parameters**

|    | name | The new name for the RecordStore. |

**Exceptions**

- **Error::StrategyError**: An error occurred when using the underlying storage system, or the name is malformed.

Reimplemented in BiometricEvaluation::IO::FileRecordStore.

E.27.3.5  virtual void BiometricEvaluation::IO::RecordStore::changeDescription ( const string & description ) throw (Error::StrategyError) [virtual]

Change the description of the RecordStore.

**Parameters**

|    | description | The new description. |

**Exceptions**

- **Error::StrategyError**: An error occurred when using the underlying storage system.

E.27.3.6  virtual uint64_t BiometricEvaluation::IO::RecordStore::getSpaceUsed ( ) const throw (Error::StrategyError) [virtual]

Obtain real storage utilization.

The amount of disk space used, for example. This is the actual space allocated by the underlying storage mechanism, in bytes.

**Returns**
The amount of backing storage used by the RecordStore.
Exceptions

| Error::StrategyError | An error occurred when using the underlying storage system. |

Reimplemented in BiometricEvaluation::IO::FileRecordStore.

E.27.3.7 virtual void BiometricEvaluation::IO::RecordStore::sync ( ) const throw (Error::StrategyError) [virtual]

Synchronize the entire record store to persistent storage.

Exceptions

| Error::StrategyError | An error occurred when using the underlying storage system. |

E.27.3.8 virtual void BiometricEvaluation::IO::RecordStore::insert ( const string & key, const void * const data, const uint64_t size ) throw (Error::ObjectExists, Error::StrategyError) [pure virtual]

Insert a record into the store.

Parameters

| in  | key   | The key of the record to be inserted. |
| in  | data  | The data for the record.              |
| in  | size  | The size, in bytes, of the record.    |

Exceptions

| Error::ObjectExists | A record with the given key is already present. |
| Error::StrategyError | An error occurred when using the underlying storage system. |

Implemented in BiometricEvaluation::IO::FileRecordStore.

E.27.3.9 virtual void BiometricEvaluation::IO::RecordStore::insert ( const string & key, const Memory::uint8Array & data ) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Insert a record into the store.

Parameters

| in  | key   | The key of the record to be inserted. |
| in  | data  | The data for the record.              |

Exceptions

| Error::ObjectExists | A record with the given key is already present. |
| Error::StrategyError | An error occurred when using the underlying storage system. |
E.27.3.10 virtual void BiometricEvaluation::IO::RecordStore::remove ( const string & key ) throw ( Error::ObjectDoesNotExist, Error::StrategyError ) [pure virtual]

Remove a record from the store.

Parameters

| in   | key | The key of the record to be removed. |

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError     | An error occurred when using the underlying storage system. |

Implemented in BiometricEvaluation::IO::FileRecordStore.

E.27.3.11 virtual uint64_t BiometricEvaluation::IO::RecordStore::read ( const string & key, void *const data ) const throw ( Error::ObjectDoesNotExist, Error::StrategyError ) [pure virtual]

Read a complete record from a store. Applications are responsible for allocating storage for the record’s data.

Parameters

| in   | key | The key of the record to be read. |
| in   | data | Pointer to where the data is to be written. |

Returns

The size of the record.

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError     | An error occurred when using the underlying storage system. |

Implemented in BiometricEvaluation::IO::FileRecordStore.

E.27.3.12 virtual uint64_t BiometricEvaluation::IO::RecordStore::read ( const string & key, Memory::uint8Array & data ) const throw ( Error::ObjectDoesNotExist, Error::StrategyError ) [virtual]

Read a complete record from a store. The AutoArray will be resized to match the size of the data.

Parameters

| in   | key | The key of the record to be read. |
| in   | data | Pointer to where the data is to be written. |

Returns

The size of the record.

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError     | An error occurred when using the underlying storage system. |
virtual void BiometricEvaluation::IO::RecordStore::replace ( const string & key, const void * const data, const uint64_t size ) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

Replace a complete record in a store.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>The key of the record to be replaced.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>The data for the record.</td>
</tr>
<tr>
<td>in</td>
<td>size</td>
<td>The size of data.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Error::ObjectDoesNotExist</th>
<th>A record for the key does not exist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

Implemented in BiometricEvaluation::IO::FileRecordStore.

virtual void BiometricEvaluation::IO::RecordStore::replace ( const string & key, const Memory::uint8Array & data ) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Replace a complete record in a RecordStore.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>The key of the record to be replaced.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>data</td>
<td>The data for the record.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Error::ObjectDoesNotExist</th>
<th>A record for the key does not exist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

virtual uint64_t BiometricEvaluation::IO::RecordStore::length ( const string & key ) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

Return the length of a record.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>key</th>
<th>The key of the record.</th>
</tr>
</thead>
</table>

Returns

The record length.

Exceptions

<table>
<thead>
<tr>
<th>Error::ObjectDoesNotExist</th>
<th>A record for the key does not exist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

Implemented in BiometricEvaluation::IO::FileRecordStore.
Commit the record’s data to storage.

Parameters

| in  | key | The key of the record to be flushed. |

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError      | An error occurred when using the underlying storage system. |

Implemented in BiometricEvaluation::IO::FileRecordStore.

Sequence through a RecordStore, returning the key/data pairs.

Parameters

| out | key | The key of the currently sequenced record. |
| in  | data | Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted. |
| in  | cursor | The location within the sequence of the key/data pair to return. |

Returns
The length of the record currently in sequence.

Exceptions

| Error::ObjectDoesNotExist | A record for the key does not exist. |
| Error::StrategyError      | An error occurred when using the underlying storage system. |

Implemented in BiometricEvaluation::IO::FileRecordStore.
out | key | The key of the currently sequenced record.
in | data | Pointer to where the data is to be written.
in | cursor | The location within the sequence of the key/data pair to return.

Returns
The length of the record currently in sequence.

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::ObjectDoesNotExist</td>
<td>A record for the key does not exist.</td>
</tr>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

E.27.3.19 virtual void BiometricEvaluation::IO::RecordStore::setCursorAtKey (string & key) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

Set the sequence cursor to an arbitrary position within the RecordStore, starting at key. Key will be the first record returned from the next call to sequence().

Parameters

in | key | The key of the record which will be returned by the first subsequent call to sequence().

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::ObjectDoesNotExist</td>
<td>A record for the key does not exist.</td>
</tr>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

Implemented in BiometricEvaluation::IO::FileRecordStore.

E.27.3.20 static tr1::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::openRecordStore (const string & name, const string & parentDir, uint8_t mode = READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]

Open an existing RecordStore and return a managed pointer to the the object representing that store.

Applications can open existing record stores without the need to know what type of RecordStore it is.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

in | name | The name of the store to be opened. |
in | parentDir | Where, in the file system, the store is rooted. |
in | mode | The type of access a client of this RecordStore has. |

Returns
An object representing the existing store.

Exceptions

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::ObjectDoesNotExist</td>
<td>The RecordStore does not exist.</td>
</tr>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system, or the name is malformed.</td>
</tr>
</tbody>
</table>

Generated on July 3, 2012
Create a new `RecordStore` and return a managed pointer to the object representing that store.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>The name of the store to be created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>description</td>
<td>The description of the store to be created.</td>
</tr>
<tr>
<td>in</td>
<td>type</td>
<td>The type of the store to be created.</td>
</tr>
<tr>
<td>in</td>
<td>destDir</td>
<td>Where, in the file system, the store will be created.</td>
</tr>
</tbody>
</table>

**Returns**

An auto_ptr to the object representing the created store.

**Exceptions**

- `Error::ObjectDoesNotExist` The `RecordStore` does not exist.
- `Error::StrategyError` An error occurred when using the underlying storage system, or the name is malformed.

Remove a `RecordStore` by deleting all persistent data associated with the store.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>name</th>
<th>The name of the existing <code>RecordStore</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>parentDir</td>
<td>Where, in the file system, the store is rooted.</td>
</tr>
</tbody>
</table>

**Exceptions**

- `Error::ObjectDoesNotExist` A record with the given key does not exist.
- `Error::StrategyError` An error occurred when using the underlying storage system.

Create a new `RecordStore` that contains the contents of several `RecordStores`.

**Parameters**

<table>
<thead>
<tr>
<th>in</th>
<th>mergedName</th>
<th>The name of the new <code>RecordStore</code> that will be created.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>mergedDescription</td>
<td>The text used to describe the <code>RecordStore</code>.</td>
</tr>
<tr>
<td>in</td>
<td>parentDir</td>
<td>Where, in the file system, the new store should be rooted.</td>
</tr>
<tr>
<td>in</td>
<td>type</td>
<td>The type of <code>RecordStore</code> that mergedName should be.</td>
</tr>
<tr>
<td>in</td>
<td>recordStores</td>
<td>An array of <code>RecordStore*</code> that should be merged into mergedName.</td>
</tr>
<tr>
<td>in</td>
<td>numRecordStores</td>
<td>The number of <code>RecordStore*</code> in recordStores.</td>
</tr>
</tbody>
</table>
E.27 BiometricEvaluation::IO::RecordStore Class Reference

Exceptions

<table>
<thead>
<tr>
<th>Error::ObjectExists</th>
<th>A RecordStore with mergedNamed in parentDir already exists.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

E.27.3.24 static void BiometricEvaluation::IO::RecordStore::mergeRecordStores ( const string & mergedName, const string & mergedDescription, const string & parentDir, const string & type, tr1::shared_ptr< RecordStore > recordStores[], size_t numRecordStores ) throw (Error::ObjectExists, Error::StrategyError) [static]

Create a new RecordStore that contains the contents of several RecordStores.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mergedName</td>
<td>The name of the new RecordStore that will be created.</td>
</tr>
<tr>
<td>mergedDescription</td>
<td>The text used to describe the RecordStore.</td>
</tr>
<tr>
<td>parentDir</td>
<td>Where, in the file system, the new store should be rooted.</td>
</tr>
<tr>
<td>type</td>
<td>The type of RecordStore that mergedName should be.</td>
</tr>
<tr>
<td>recordStores</td>
<td>An array of RecordStore shared pointers, such as those returned from IO::Factory, that should be merged into mergedName.</td>
</tr>
<tr>
<td>numRecordStores</td>
<td>The number of RecordStore* in recordStores.</td>
</tr>
</tbody>
</table>

Exceptions

<table>
<thead>
<tr>
<th>Error::ObjectExists</th>
<th>A RecordStore with mergedNamed in parentDir already exists.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error::StrategyError</td>
<td>An error occurred when using the underlying storage system.</td>
</tr>
</tbody>
</table>

E.27.3.25 static string BiometricEvaluation::IO::RecordStore::genKeySegName ( const string & key, const uint64_t segnum ) [static, protected]

Generate key segment names.

Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>Base key name.</td>
</tr>
<tr>
<td>segnum</td>
<td>Segment number for key (zero based).</td>
</tr>
</tbody>
</table>

Returns

Key segment name.

E.27.4 Member Data Documentation

E.27.4.1 const string BiometricEvaluation::IO::RecordStore::INVALIDKEYCHARS [static]

The set of prohibited characters in a key: '/', '\', '*', '&' 

E.27.4.2 const char BiometricEvaluation::IO::RecordStore::KEY_SEGMENT_SEPARATOR = '&' [static]

Character used to separate key segments
E.27.4.3  const uint64_t BiometricEvaluation::IO::RecordStore::KEY_SEGMENT_START = 1 [static]

First segment number of a segmented record

E.27.4.4  const string BiometricEvaluation::IO::RecordStore::CONTROLFILENAME [static]

The name of the control file, a properties list

E.27.4.5  const string BiometricEvaluation::IO::RecordStore::NAMEPROPERTY [static]

Property key for name of the RecordStore

E.27.4.6  const string BiometricEvaluation::IO::RecordStore::DESCRIPTIONPROPERTY [static]

Property key for description of the RecordStore

E.27.4.7  const string BiometricEvaluation::IO::RecordStore::COUNTPROPERTY [static]

Property key for the number of store items

E.27.4.8  const string BiometricEvaluation::IO::RecordStore::TYPEPROPERTY [static]

Property key for the type of RecordStore

E.27.4.9  const string BiometricEvaluation::IO::RecordStore::BERKELEYDBTYPE [static]

DBRecordStore type

E.27.4.10 const string BiometricEvaluation::IO::RecordStore::ARCHIVETYPE [static]

ArchiveRecordStore type

E.27.4.11 const string BiometricEvaluation::IO::RecordStore::FILETYPE [static]

FileRecordStore type

E.27.4.12 const string BiometricEvaluation::IO::RecordStore::SQLITETYPE [static]

SQLiteRecordStore type

E.27.4.13 const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_START = 1 [static]

Tell sequence() to sequence from beginning
### E.28 BiometricEvaluation::Image::Resolution Struct Reference

A structure to represent the resolution of an image.

```cpp
#include <be_image.h>
```

#### Public Types

- `enum Kind { NA = 0, PPI = 1, PPMM = 2, PPCM = 3 }`

    Possible representations of the units in a `Resolution` struct.

#### Public Member Functions

- `Resolution (const double xRes=0.0, const double yRes=0.0, const Kind units=PPI)`

    Create a `Resolution` struct.

#### Public Attributes

- `double xRes`
- `double yRes`
- `Kind units`

### E.28.1 Detailed Description

A structure to represent the resolution of an image.

### E.28.2 Member Enumeration Documentation

#### E.28.2.1 enum BiometricEvaluation::Image::Resolution::Kind

Possible representations of the units in a `Resolution` struct.

**Enumerator:**

- `NA` Not-applicable: unknown, or otherwise
- `PPI` Pixels per inch
- `PPMM` Pixels per millimeter
- `PPCM` Pixels per centimeter
E.28.3 Constructor & Destructor Documentation

E.28.3.1 BiometricEvaluation::Image::Resolution::Resolution ( const double \textit{xRes} = 0.0, const double \textit{yRes} = 0.0, const Kind \textit{units} = PPI )

Create a Resolution struct.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{xRes}</td>
<td>Resolution along the X-axis</td>
</tr>
<tr>
<td>\textit{yRes}</td>
<td>Resolution along the Y-axis</td>
</tr>
<tr>
<td>\textit{units}</td>
<td>Units in which \textit{xRes} and \textit{yRes} are represented</td>
</tr>
</tbody>
</table>

E.28.4 Member Data Documentation

E.28.4.1 double BiometricEvaluation::Image::Resolution::xRes

Resolution along the X-axis

E.28.4.2 double BiometricEvaluation::Image::Resolution::yRes

Resolution along the Y-axis

E.28.4.3 Kind BiometricEvaluation::Image::Resolution::units

Units in which \textit{xRes} and \textit{yRes} are represented

The documentation for this struct was generated from the following file:

- be_image.h

E.29 FPVTE2012::ReturnStatus Struct Reference

A structure to contain information about a failure by the software under test.

#include <fpvte2012.h>

Public Member Functions

- \textbf{ReturnStatus} (const StatusCode::Kind \textit{code}, const string \textit{info}="")
  
  Create a ReturnStatus object.

Public Attributes

- StatusCode::Kind \textit{code}
- string \textit{info}
E.29.1 Detailed Description

A structure to contain information about a failure by the software under test.

An object of this class allows the software to return some information from a function call. The string within this object can be optionally set to provide more information for debugging etc. The status code will be set by the function to Success on success, or one of the other codes on failure. In those cases, processing will proceed with further calls to the function.

Note

If the SDK encounters a non-recoverable error, an exception should be thrown and processing will stop.

E.29.2 Constructor & Destructor Documentation

E.29.2.1 FPVTE2012::ReturnStatus::ReturnStatus ( const StatusCode::Kind code, const string info = "" )

Create a ReturnStatus object.

Parameters

<table>
<thead>
<tr>
<th>in</th>
<th>code</th>
<th>The return status code; required.</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>info</td>
<td>The optional information string.</td>
</tr>
</tbody>
</table>

The documentation for this struct was generated from the following file:

- fpvte2012.h

E.30 FPVTE2012::SDKInterface Class Reference

Public Member Functions

- virtual void getIDs (string &nist_assigned_identifier, string &email_address)=0

  Return the identifier and contact email address for the software under test.

- virtual ReturnStatus initEnrollmentTemplateExtraction (const string &configuration_directory, const uint32_t num_subjects)=0

  This function initializes the makeEnrollmentTemplate process and sets all needed parameters.

- virtual ReturnStatus makeEnrollmentTemplate (const FingerImageSet &fingers, Memory::uint8Array &enrollment_template, FingerOutputSet &output_features)=0 throw (Error::Exception)

  This function takes a FingerImageSet and outputs a enrollment template representing the one or more fingers that were in the input FingerImageSet.

- virtual ReturnStatus finalizeEnrollment (const string &configuration_directory, const string &enrollment_directory, const uint8_t blade_count, const uint16_t blade_memory, IO::RecordStore &input_templates)=0 throw (Error::Exception)

  This function does final processing of the complete set of enrollment templates that will then be used in matching search templates.

- virtual ReturnStatus initSearchTemplateExtraction (const string &configuration_directory)=0 throw (Error::Exception)

  This function initializes the makeSearchTemplate process and sets all needed parameters.

- virtual ReturnStatus makeSearchTemplate (const FingerImageSet &fingers, Memory::uint8Array &search_template, FingerOutputSet &output_features)=0 throw (Error::Exception)

  This function takes a FingerImageSet and outputs a search template representing one or more fingers. The search template can be produced with a different algorithm than makeEnrollmentTemplate.

- virtual ReturnStatus initIdentificationStageOne (const string &configuration_directory, const string &enrollment_directory, const uint8_t blade_number)=0 throw (Error::Exception)
This function initializes identifyTemplateStageOne.

- virtual ReturnStatus identifyTemplateStageOne (const string &search_ID, const Memory::uint8Array &search_template, const string &stage1_data_directory)=0 throw (Error::Exception)

This function searches a template against the partial enrollment set selected by the blade_number in initIdentificationStageOne.

- virtual ReturnStatus initIdentificationStageTwo (const string &configuration_directory, const string &enrollment_directory)=0 throw (Error::Exception)

This function initializes identifyTemplateStageTwo.

- virtual ReturnStatus identifyTemplateStageTwo (const string &search_ID, const string &stage1_data_directory, CandidateSet &candidates)=0 throw (Error::Exception)

This function takes the results from identifyTemplateStageOne and produces a candidate list for the search subject.

Static Public Member Functions

- static SDKIptr getSDK ()

Factory function to return a managed pointer to the SDK object.

E.30.1 Member Function Documentation

E.30.1.1 virtual void FPVTE2012::SDKInterface::getIDs ( string &nist_assigned_identifier, string &email_address ) [pure virtual]

Return the identifier and contact email address for the software under test.

This function retrieves an identifier that the provider must request from NIST and hard wired into the source code. NIST will assign the identifier that will uniquely identify the supplier and the SDK version number.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out nist_assigned_identifier</td>
<td>This string will be the full name of the core library file. For example, if the NIST assigned identifier is &quot;A&quot;, the class of participation is &quot;C&quot; and the SDK is the first &quot;fast&quot; * submission, the library name AND string returned by getIDs would * be libFpVTE12_A_C_1_1.a</td>
</tr>
<tr>
<td>out email_address</td>
<td>Point of contact email address. Cannot be the empty string.</td>
</tr>
</tbody>
</table>

E.30.1.2 virtual ReturnStatus FPVTE2012::SDKInterface::initEnrollmentTemplateExtraction ( const string &configuration_directory, const uint32 num_subjects ) throw (Error::Exception) [pure virtual]

This function initializes the makeEnrollmentTemplate process and sets all needed parameters.

The function is called once by the NIST application before n >= 1 calls to makeEnrollmentTemplate. The implementation must tolerate execution of multiple instances of the makeEnrollmentTemplate process (each initialized separately) running simultaneously and independently on the same machine and/or across multiple machines.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in configuration_directory</td>
<td>A read-only directory containing vendor-supplied configuration parameters or run-time data files.</td>
</tr>
<tr>
<td>in num_subjects</td>
<td>The total number of subjects that will be used in the enrollment set.</td>
</tr>
</tbody>
</table>

Returns

The object containing a required status code and optional information string.

Exceptions

Error::Exception | There was an error processing this request, and the exception string may contain additional information.
virtual ReturnStatus FPVTE2012::SDKInterface::makeEnrollmentTemplate ( const FingerImageSet & fingers, Memory::uint8Array & enrollment_template, FingerOutputSet & output_features ) throw (Error::Exception) [pure virtual]

This function takes a FingerImageSet and outputs a enrollment template representing the one or more fingers that were in the input FingerImageSet.

The memory for the template is managed in the uint8Array (an AutoArray) object. This routine should not perform memory allocation but can resize the AutoArray if more space is needed.

**Note**

This method should call AutoArray’s resize() method before returning in order to indicate the true size of enrollment_template.

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>fingers</td>
<td>A set of finger images. The implementation will have to adjust to the number of available fingerprints based on the dataset being used. For example 10 rolled or 2 index plain impressions.</td>
</tr>
<tr>
<td>out</td>
<td>enrollment_template</td>
<td>The template to be added to the enrollment set, extracted from the finger images. The format is not regulated.</td>
</tr>
<tr>
<td>out</td>
<td>output_features</td>
<td>Additional information for the fingers, produced by the SDK. This information will be used in final results analysis. The number of features can be different than the number of finger images, when segmentation is done, for example.</td>
</tr>
</tbody>
</table>

### Returns

The object containing a required status code and optional information string.

### Exceptions

**Error::Exception**  
There was an error processing this request, and the exception string may contain additional information.

virtual ReturnStatus FPVTE2012::SDKInterface::finalizeEnrollment ( const string & configuration_directory, const string & enrollment_directory, const uint8_t blade_count, const uint16_t blade_memory, IO::RecordStore & input_templates ) throw (Error::Exception) [pure virtual]

This function does final processing of the complete set of enrollment templates that will then be used in matching search templates.

The finalization step must prepare the enrolled templates to be distributed across a number of blades. The enrollment directory will then have read-only access for stages 1 and 2 of the identification process.

### Parameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>configuration_directory</td>
<td>A read-only directory containing vendor-supplied configuration</td>
</tr>
<tr>
<td>in</td>
<td>enrollment_directory</td>
<td>The top-level directory in which all enrollment data will reside. After this finalization step the directory and its contents with become read-only. The directory is initially empty. Access permission will be read-write and the application can populate this folder as needed. Additionally, the file system does not perform well with the creation of millions of small files, so the application should consolidate templates into some sort of database file.</td>
</tr>
<tr>
<td>in</td>
<td>blade_count</td>
<td>The number of blades the enrollment set will be spread across. It is up to the implementation to determine how best to spread the enrolled templates across the blades in order to get best performance.</td>
</tr>
<tr>
<td>in</td>
<td>blade_memory</td>
<td>Amount of memory available on each blade in gigabytes.</td>
</tr>
<tr>
<td>in</td>
<td>input_templates</td>
<td>The input RecordStore containing the enrolled templates, opened IO::READONLY.</td>
</tr>
</tbody>
</table>
Returns
The object containing a required status code and optional information string.

Exceptions

| Error::Exception | There was an error processing this request, and the exception string may contain additional information. |

E.30.1.5 virtual ReturnStatus FPVTE2012::SDKInterface::initSearchTemplateExtraction (const string &configuration_directory) throw (Error::Exception) [pure virtual]

This function initializes the makeSearchTemplate process and sets all needed parameters.

The function is called once by the NIST application before \( n \geq 1 \) calls to makeSearchTemplate. The implementation must tolerate execution of multiple instances of the makeSearchTemplate process running simultaneously and independently on the same machine and/or across multiple machines.

Parameters

| in | configuration_directory | A read-only directory containing vendor-supplied configuration |

Returns
The object containing a required status code and optional information string.

Exceptions

| Error::Exception | There was an error processing this request, and the exception string may contain additional information. |

E.30.1.6 virtual ReturnStatusFPVTE2012::SDKInterface::makeSearchTemplate (const FingerImageSet &fingers, Memory::uint8Array &search_template, FingerOutputSet &output_features) throw (Error::Exception) [pure virtual]

This function takes a FingerImageSet and outputs a search template representing one or more fingers. The search template can be produced with a different algorithm than makeEnrollmentTemplate.

The memory for the template is managed in the uint8Array (an AutoArray) object. This routine should not perform memory allocation but can resize the AutoArray if more space is needed.

Note

This method should call AutoArray’s resize() method before returning in order to indicate the true size of search_template.

Parameters

| in | fingers | A set of finger images. The implementation will have to adjust to the number of available fingerprints based on the dataset being used. For example 10 rolled or 2 index plain impressions. |
| in | fingers | A set of finger images. The implementation will have to adjust to the number of available fingerprints based on the dataset being used. For example 10 rolled or 2 index plain impressions. |
| out | search_template | The search template, extracted from the finger images. The format is not regulated. The memory is allocated in the AutoArray object. |
| out | output_features | Additional information for the fingers, produced by the SDK. This information will be used in final results analysis. The number of features can be different than the number of finger images, when segmentation is done, for example. |
Returns
The object containing a required status code and optional information string.

Exceptions

| Error::Exception | There was an error processing this request, and the exception string may contain additional information. |

E.30.1.7 virtual ReturnStatus FPVTE2012::SDKInterface::initIdentificationStageOne ( const string & configuration_directory, const string & enrollment_directory, const uint8_t blade_number ) throw (Error::Exception) [pure virtual]

This function initializes identifyTemplateStageOne.
The function will be called to initialize each blade that will contain a portion of the enrolled templates. The number of blades will be the same as used in finalizeEnrollment.

Parameters

| in | configuration_directory | A read-only directory containing vendor-supplied configuration |
| in | enrollment_directory | The top-level directory in which all finalized enrollment data resides. The directory will have read-only access. |
| in | blade_number | Blade number from blades in finalizeEnrollment that is being initialized. This parameter lets the process know which piece of the enrolled templates to load into memory. Blades are numbered 0 to B - 1. |

Returns
The object containing a required status code and optional information string.

Exceptions

| Error::Exception | There was an error processing this request, and the exception string may contain additional information. |

E.30.1.8 virtual ReturnStatus FPVTE2012::SDKInterface::identifyTemplateStageOne ( const string & search_ID, const Memory::uint8Array & search_template, const string & stage1_data_directory ) throw (Error::Exception) [pure virtual]

This function searches a template against the partial enrollment set selected by the blade_number in initIdentificationstageOne.

Parameters

| in | search_ID | The ID of the search template. This ID does not identify subject, but is merely an identifier used to distinguish different searches performed by the system. It will be used as the input to identifyTemplateStageTwo. |
| in | search_template | A template from makeSearchTemplate. The size is given by the AutoArray object. |
| in | stage1_data_directory | This directory will have read-write access. The output information from identifyTemplateStageOne that is needed in identifyTemplateStageTwo is written in this directory. This directory will be unique for each search performed. |

Returns
The object containing a required status code and optional information string.
Exceptions

| Error::Exception | There was an error processing this request, and the exception string may contain additional information. |

E.30.1.9 virtual ReturnStatus FPVTE2012::SDKInterface::initIdentificationStageTwo ( const string & configuration_directory, const string & enrollment_directory ) throw (Error::Exception) [pure virtual]

This function initializes identifyTemplateStageTwo.

This second stage of identification uses the output results from identifyTemplateStageOne to produce a candidate list for the search subject.

Parameters

| in  | configuration_directory | A read-only directory containing vendor-supplied configuration |
| in  | enrollment_directory   | The top-level directory in which all finalized enrolled data resides. The directory will have read-only access. |

Returns

The object containing a required status code and optional information string.

Exceptions

| Error::Exception | There was an error processing this request, and the exception string may contain additional information. |

E.30.1.10 virtual ReturnStatus FPVTE2012::SDKInterface::identifyTemplateStageTwo ( const string & search_ID, const string & stage1_data_directory, CandidateSet & candidates ) throw (Error::Exception) [pure virtual]

This function takes the results from identifyTemplateStageOne and produces a candidate list for the search subject.

Parameters

| in   | search_ID             | The ID of the search template. This ID does not identify subject, but is merely an identifier used to distinguish different searches performed by the system. |
| in   | stage1_data_directory | This directory will have read-only access and contains all the stored output data from the identifyTemplateStageOne processes for the search subject. |
| out  | candidates            | The candidate list returned from identifyTemplateStageTwo. |

Returns

The object containing a required status code and optional information string.

Exceptions

| Error::Exception | There was an error processing this request, and the exception string may contain additional information. |
Factory function to return a managed pointer to the SDK object.
This function is implemented by the SDK library and must return a managed pointer to the SDK object.
The documentation for this class was generated from the following file:

- fpvte2012.h

### E.31 BiometricEvaluation::Image::Size Struct Reference

A structure to represent the size of an image, in pixels.

```
#include <be_image.h>
```

#### Public Member Functions

- `Size (const uint32_t xSize=0, const uint32_t ySize=0)`
  
  Create a Size struct.

#### Public Attributes

- `uint32_t xSize`
- `uint32_t ySize`

#### E.31.1 Detailed Description

A structure to represent the size of an image, in pixels.

#### E.31.2 Constructor & Destructor Documentation

E.31.2.1 `BiometricEvaluation::Image::Size::Size ( const uint32_t xSize = 0, const uint32_t ySize = 0 )`

Create a Size struct.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xSize</code></td>
<td>Number of pixels on the X-axis</td>
</tr>
<tr>
<td><code>ySize</code></td>
<td>Number of pixels on the Y-axis</td>
</tr>
</tbody>
</table>

#### E.31.3 Member Data Documentation

E.31.3.1 `uint32_t BiometricEvaluation::Image::Size::xSize`

Number of pixels on the X-axis
E.31.3.2 uint32 BiometricEvaluation::Image::Size::ySize

Number of pixels on the Y-axis

The documentation for this struct was generated from the following file:

- be_image.h

E.32 FPVTE2012::StatusCode Class Reference

Public Types

- enum Kind { Success = 0, ImageSizeNotSupported = 1, TemplateTypeNotSupported = 2, FailedToExtract = 3, FailedToMatch = 4, FailedToParseInput = 5, Vendor = 6 }

A class that contains an enumeration which defines the set of status codes.

E.32.1 Member Enumeration Documentation

E.32.1.1 enum FPVTE2012::StatusCode::Kind

A class that contains an enumeration which defines the set of status codes.

The status codes that are returned from a function call:

Enumerator:

- **Success**  
  Successful completion

- **ImageSizeNotSupported**  
  Image size too small or large

- **TemplateTypeNotSupported**  
  Unsupported template type

- **FailedToExtract**  
  Could not extract template from image

- **FailedToMatch**  
  Could not match samples

- **Vendor**  
  Vendor-defined error

The documentation for this class was generated from the following file:

- fpvte2012.h

E.33 BiometricEvaluation::Error::StrategyError Class Reference

A *StrategyError* object is thrown when the underlying implementation of this interface encounters an error.

```
#include <be_error_exception.h>
```

Inheritance diagram for BiometricEvaluation::Error::StrategyError:

```
BiometricEvaluation::Error::Exception
|---------------------|
| BiometricEvaluation::Error::StrategyError
```
Public Member Functions

- `StrategyError()`
- `StrategyError(string info)`

E.33.1 Detailed Description

A `StrategyError` object is thrown when the underlying implementation of this interface encounters an error.

E.33.2 Constructor & Destructor Documentation

E.33.2.1 BiometricEvaluation::Error::StrategyError::StrategyError()

Construct a `StrategyError` object with the default information string.

**Returns**
The `StrategyError` object.

E.33.2.2 BiometricEvaluation::Error::StrategyError::StrategyError(string info)

Construct a `StrategyError` object with an information string appended to the default information string.

**Returns**
The `StrategyError` object.

The documentation for this class was generated from the following file:

- `be_error_exception.h`