

FpVTE 2012  
FINGERPRINT VENDOR TECHNOLOGY EVALUATION  
IMPLEMENTER'S GUIDE  
FINAL VERSION 0.3

---

CRAIG WATSON  
WAYNE SALAMON  
GREGORY FIUMARA

IMAGE GROUP  
INFORMATION ACCESS DIVISION  
INFORMATION TECHNOLOGY LABORATORY

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

MAY 18, 2012

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background	1
1.2	Purpose	1
1.3	Audience	2
1.4	Application Scenarios	2
1.5	Timeline	4
1.6	Offline Testing	4
1.7	Phased Testing	4
1.7.1	Interim Reports	4
1.7.2	Publication of results	5
1.8	Reporting of Failure to Enroll or Acquire (FTE, FTA)	5
1.9	Matching of Empty, Broken, and Missing Templates	5
1.10	Reporting of Template Size	5
1.11	Runtime Memory Usage	6
1.12	Reporting of Computational Efficiency	6
1.13	Exploring the Accuracy-Speed Trade-Off	6
1.14	Hardware Specification	6
1.15	Operating system and Compilation Environment	6
1.16	Threaded Computations	7
1.17	Time Limits	7
1.18	Ground Truth Integrity	7
<b>2</b>	<b>Datasets</b>	<b>9</b>
2.1	Image Types	9
<b>3</b>	<b>SDK Application Programming Interface</b>	<b>10</b>
3.1	1:N Identification with Large Enrollment Set	10
3.1.1	Threading and Multi-Processing	11
3.1.2	Implementation Identifiers	11
3.2	Input Fingerprint Image Structure	11
3.3	Output Fingerprint Template Structure	11
3.3.1	Enrollment Template Extraction and Finalization	12
3.3.1.1	Enrollment Template Extraction	12
3.3.1.2	Enrollment Finalization	12
3.3.2	Search Template Extraction	13
3.3.3	Identification Stage One	15
3.4	Output Candidate Lists Structure	15
3.4.1	Identification Stage Two	15
3.5	SDK C++ Interface	16
3.5.1	The SDK Interface	16
3.5.2	Exceptions	18
3.5.3	Biometric Evaluation Classes	18
3.5.3.1	IO::RecordStore	19
3.5.3.2	Memory::AutoArray	19

3.5.3.3	Error::Exception	19
<b>4</b>	<b>Software and Documentation</b>	<b>20</b>
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
<b>5</b>	<b>How to Submit Implementations to FpVTE2012</b>	<b>23</b>
5.1	Confidentiality and Integrity Protection	23
5.2	How to Participate	23
5.3	Implementation Validation	24
<b>A</b>	<b>Namespace Index</b>	<b>26</b>
A.1	Namespace List	26
<b>B</b>	<b>Class Index</b>	<b>27</b>
B.1	Class Hierarchy	27
<b>C</b>	<b>Class Index</b>	<b>28</b>
C.1	Class List	28
<b>D</b>	<b>Namespace Documentation</b>	<b>30</b>
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	Variable Documentation	34
D.4.2.1	READWRITE	34
D.4.2.2	READONLY	34
D.5	BiometricEvaluation::IO::Utility Namespace Reference	34
D.5.1	Detailed Description	34
D.5.2	Function Documentation	34
D.5.2.1	removeDirectory	34
D.5.2.2	getFileSize	35
D.5.2.3	fileExists	35
D.5.2.4	validateRootName	35
D.5.2.5	constructAndCheckPath	36
D.6	BiometricEvaluation::Text Namespace Reference	36

D.6.1	Detailed Description	36
D.7	FPVTE2012 Namespace Reference	36
D.7.1	Detailed Description	37
D.7.2	Typedef Documentation	37
D.7.2.1	FingerImageSet	37
D.7.2.2	FingerOutputSet	37
D.7.2.3	CandidateSet	37
D.7.3	Function Documentation	37
D.7.3.1	operator<<	37
D.7.3.2	operator<<	38
<b>E</b>	<b>Class Documentation</b>	<b>39</b>
E.1	BiometricEvaluation::Memory::AutoArray< T > Class Template Reference	39
E.1.1	Detailed Description	40
E.1.2	Member Typedef Documentation	40
E.1.2.1	value_type	40
E.1.2.2	size_type	40
E.1.2.3	iterator	40
E.1.2.4	const_iterator	40
E.1.2.5	reference	40
E.1.2.6	const_reference	41
E.1.3	Constructor & Destructor Documentation	41
E.1.3.1	AutoArray	41
E.1.3.2	AutoArray	41
E.1.3.3	AutoArray	41
E.1.3.4	~AutoArray	41
E.1.4	Member Function Documentation	41
E.1.4.1	operator T *	41
E.1.4.2	operator const T *	41
E.1.4.3	operator[]	42
E.1.4.4	operator[]	42
E.1.4.5	at	42
E.1.4.6	at	42
E.1.4.7	begin	43
E.1.4.8	begin	43
E.1.4.9	end	43
E.1.4.10	end	43
E.1.4.11	size	44
E.1.4.12	resize	44
E.1.4.13	copy	44
E.1.4.14	copy	44
E.1.4.15	operator=	45
E.2	FPVTE2012::Candidate Struct Reference	45
E.2.1	Detailed Description	45
E.3	BiometricEvaluation::Image::CompressionAlgorithm Class Reference	45
E.3.1	Detailed Description	46
E.4	BiometricEvaluation::Error::ConversionError Class Reference	46
E.4.1	Detailed Description	46
E.4.2	Constructor & Destructor Documentation	46
E.4.2.1	ConversionError	46
E.4.2.2	ConversionError	46
E.5	BiometricEvaluation::Image::Coordinate Struct Reference	47
E.5.1	Detailed Description	47
E.5.2	Constructor & Destructor Documentation	47
E.5.2.1	Coordinate	47

E.5.3	Member Data Documentation	47
E.5.3.1	x	47
E.5.3.2	y	48
E.5.3.3	xDistance	48
E.5.3.4	yDistance	48
E.6	BiometricEvaluation::Error::DataError Class Reference	48
E.6.1	Detailed Description	48
E.6.2	Constructor & Destructor Documentation	48
E.6.2.1	DataError	48
E.6.2.2	DataError	49
E.7	BiometricEvaluation::Error::Exception Class Reference	49
E.7.1	Detailed Description	50
E.7.2	Constructor & Destructor Documentation	50
E.7.2.1	Exception	50
E.7.2.2	Exception	50
E.7.3	Member Function Documentation	50
E.7.3.1	getInfo	50
E.8	BiometricEvaluation::Error::FileError Class Reference	51
E.8.1	Detailed Description	51
E.8.2	Constructor & Destructor Documentation	51
E.8.2.1	FileError	51
E.8.2.2	FileError	51
E.9	BiometricEvaluation::IO::FileRecordStore Class Reference	51
E.9.1	Detailed Description	52
E.9.2	Constructor & Destructor Documentation	52
E.9.2.1	FileRecordStore	52
E.9.2.2	FileRecordStore	53
E.9.3	Member Function Documentation	53
E.9.3.1	getSpaceUsed	53
E.9.3.2	insert	53
E.9.3.3	remove	54
E.9.3.4	read	54
E.9.3.5	replace	54
E.9.3.6	length	55
E.9.3.7	flush	55
E.9.3.8	sequence	55
E.9.3.9	setCursorAtKey	56
E.9.3.10	changeName	56
E.10	FPVTE2012::FingerImage Struct Reference	56
E.10.1	Detailed Description	57
E.11	BiometricEvaluation::Finger::FingerImageCode Class Reference	57
E.11.1	Detailed Description	57
E.12	FPVTE2012::FingerOutput Struct Reference	57
E.12.1	Detailed Description	58
E.13	BiometricEvaluation::Image::Image Class Reference	58
E.13.1	Detailed Description	59
E.13.2	Constructor & Destructor Documentation	60
E.13.2.1	Image	60
E.13.2.2	Image	60
E.13.3	Member Function Documentation	60
E.13.3.1	getCompressionAlgorithm	60
E.13.3.2	getResolution	61
E.13.3.3	getData	61
E.13.3.4	getRawData	61

E.13.3.5	<a href="#">getRawGrayscaleData</a>	61
E.13.3.6	<a href="#">getDimensions</a>	62
E.13.3.7	<a href="#">getDepth</a>	62
E.13.3.8	<a href="#">valueInColorspace</a>	62
E.13.3.9	<a href="#">setResolution</a>	62
E.13.3.10	<a href="#">setDimensions</a>	63
E.13.3.11	<a href="#">setDepth</a>	63
E.13.4	<a href="#">Member Data Documentation</a>	63
E.13.4.1	<a href="#">bitsPerComponent</a>	63
E.13.4.2	<a href="#">_raw_data</a>	63
E.14	<a href="#">BiometricEvaluation::Finger::Impression Class Reference</a>	63
E.14.1	<a href="#">Detailed Description</a>	63
E.15	<a href="#">BiometricEvaluation::Error::MemoryError Class Reference</a>	64
E.15.1	<a href="#">Detailed Description</a>	64
E.15.2	<a href="#">Constructor &amp; Destructor Documentation</a>	64
E.15.2.1	<a href="#">MemoryError</a>	64
E.15.2.2	<a href="#">MemoryError</a>	64
E.16	<a href="#">BiometricEvaluation::Error::NotImplemented Class Reference</a>	64
E.16.1	<a href="#">Detailed Description</a>	65
E.16.2	<a href="#">Constructor &amp; Destructor Documentation</a>	65
E.16.2.1	<a href="#">NotImplemented</a>	65
E.16.2.2	<a href="#">NotImplemented</a>	65
E.17	<a href="#">BiometricEvaluation::Error::ObjectDoesNotExist Class Reference</a>	65
E.17.1	<a href="#">Detailed Description</a>	66
E.17.2	<a href="#">Constructor &amp; Destructor Documentation</a>	66
E.17.2.1	<a href="#">ObjectDoesNotExist</a>	66
E.17.2.2	<a href="#">ObjectDoesNotExist</a>	66
E.18	<a href="#">BiometricEvaluation::Error::ObjectExists Class Reference</a>	66
E.18.1	<a href="#">Detailed Description</a>	67
E.18.2	<a href="#">Constructor &amp; Destructor Documentation</a>	67
E.18.2.1	<a href="#">ObjectExists</a>	67
E.18.2.2	<a href="#">ObjectExists</a>	67
E.19	<a href="#">BiometricEvaluation::Error::ObjectIsClosed Class Reference</a>	67
E.19.1	<a href="#">Detailed Description</a>	68
E.19.2	<a href="#">Constructor &amp; Destructor Documentation</a>	68
E.19.2.1	<a href="#">ObjectIsClosed</a>	68
E.19.2.2	<a href="#">ObjectIsClosed</a>	68
E.20	<a href="#">BiometricEvaluation::Error::ObjectIsOpen Class Reference</a>	68
E.20.1	<a href="#">Detailed Description</a>	69
E.20.2	<a href="#">Constructor &amp; Destructor Documentation</a>	69
E.20.2.1	<a href="#">ObjectIsOpen</a>	69
E.20.2.2	<a href="#">ObjectIsOpen</a>	69
E.21	<a href="#">BiometricEvaluation::Error::ParameterError Class Reference</a>	69
E.21.1	<a href="#">Detailed Description</a>	70
E.21.2	<a href="#">Constructor &amp; Destructor Documentation</a>	70
E.21.2.1	<a href="#">ParameterError</a>	70
E.21.2.2	<a href="#">ParameterError</a>	70
E.22	<a href="#">BiometricEvaluation::Finger::PatternClassification Class Reference</a>	70
E.22.1	<a href="#">Detailed Description</a>	71
E.23	<a href="#">BiometricEvaluation::Finger::Position Class Reference</a>	71
E.23.1	<a href="#">Detailed Description</a>	71
E.24	<a href="#">BiometricEvaluation::IO::Properties Class Reference</a>	71
E.24.1	<a href="#">Detailed Description</a>	72
E.24.2	<a href="#">Constructor &amp; Destructor Documentation</a>	72

E.24.2.1	Properties	72
E.24.2.2	Properties	73
E.24.3	Member Function Documentation	73
E.24.3.1	setProperty	73
E.24.3.2	setPropertyFromInteger	73
E.24.3.3	setPropertyFromDouble	74
E.24.3.4	removeProperty	74
E.24.3.5	getProperty	74
E.24.3.6	getPropertyAsInteger	74
E.24.3.7	getPropertyAsDouble	75
E.24.3.8	sync	75
E.24.3.9	changeName	75
E.25	BiometricEvaluation::Image::Raw Class Reference	76
E.25.1	Detailed Description	76
E.25.2	Member Function Documentation	76
E.25.2.1	getData	76
E.25.2.2	getRawData	76
E.25.2.3	getRawGrayscaleData	77
E.26	BiometricEvaluation::IO::RecordStore Class Reference	77
E.26.1	Detailed Description	79
E.26.2	Constructor & Destructor Documentation	79
E.26.2.1	RecordStore	79
E.26.2.2	RecordStore	80
E.26.3	Member Function Documentation	80
E.26.3.1	getName	80
E.26.3.2	getDescription	80
E.26.3.3	getCount	80
E.26.3.4	changeName	80
E.26.3.5	changeDescription	81
E.26.3.6	getSpaceUsed	81
E.26.3.7	sync	81
E.26.3.8	insert	81
E.26.3.9	remove	82
E.26.3.10	read	82
E.26.3.11	replace	82
E.26.3.12	length	83
E.26.3.13	flush	83
E.26.3.14	sequence	84
E.26.3.15	setCursorAtKey	84
E.26.3.16	openRecordStore	84
E.26.3.17	createRecordStore	85
E.26.3.18	removeRecordStore	85
E.26.3.19	mergeRecordStores	86
E.26.3.20	mergeRecordStores	86
E.26.4	Member Data Documentation	86
E.26.4.1	INVALIDKEYCHARS	86
E.26.4.2	CONTROLFILENAME	86
E.26.4.3	NAMEPROPERTY	87
E.26.4.4	DESCRIPTIONPROPERTY	87
E.26.4.5	COUNTPROPERTY	87
E.26.4.6	TYPEPROPERTY	87
E.26.4.7	BERKELEYDBTYPE	87
E.26.4.8	ARCHIVETYPE	87
E.26.4.9	FILETYPE	87

E.26.4.10 SQLITETYPE . . . . .	87
E.26.4.11 BE_RECSTORE_SEQ_START . . . . .	87
E.26.4.12 BE_RECSTORE_SEQ_NEXT . . . . .	87
E.27 BiometricEvaluation::Image::Resolution Struct Reference . . . . .	88
E.27.1 Detailed Description . . . . .	88
E.27.2 Member Enumeration Documentation . . . . .	88
E.27.2.1 Kind . . . . .	88
E.27.3 Constructor & Destructor Documentation . . . . .	88
E.27.3.1 Resolution . . . . .	88
E.27.4 Member Data Documentation . . . . .	89
E.27.4.1 xRes . . . . .	89
E.27.4.2 yRes . . . . .	89
E.27.4.3 units . . . . .	89
E.28 FPVTE2012::ReturnStatus Struct Reference . . . . .	89
E.28.1 Detailed Description . . . . .	89
E.28.2 Constructor & Destructor Documentation . . . . .	89
E.28.2.1 ReturnStatus . . . . .	89
E.29 FPVTE2012::SDKInterface Class Reference . . . . .	90
E.29.1 Member Function Documentation . . . . .	91
E.29.1.1 getIDs . . . . .	91
E.29.1.2 initEnrollmentTemplateExtraction . . . . .	91
E.29.1.3 makeEnrollmentTemplate . . . . .	91
E.29.1.4 finalizeEnrollment . . . . .	92
E.29.1.5 initSearchTemplateExtraction . . . . .	92
E.29.1.6 makeSearchTemplate . . . . .	93
E.29.1.7 initIdentificationStageOne . . . . .	93
E.29.1.8 identifyTemplateStageOne . . . . .	94
E.29.1.9 initIdentificationStageTwo . . . . .	94
E.29.1.10 identifyTemplateStageTwo . . . . .	95
E.29.1.11 getSDK . . . . .	95
E.30 BiometricEvaluation::Image::Size Struct Reference . . . . .	95
E.30.1 Detailed Description . . . . .	95
E.30.2 Constructor & Destructor Documentation . . . . .	96
E.30.2.1 Size . . . . .	96
E.30.3 Member Data Documentation . . . . .	96
E.30.3.1 xSize . . . . .	96
E.30.3.2 ySize . . . . .	96
E.31 FPVTE2012::StatusCode Class Reference . . . . .	96
E.31.1 Member Enumeration Documentation . . . . .	96
E.31.1.1 Kind . . . . .	96
E.32 BiometricEvaluation::Error::StrategyError Class Reference . . . . .	97
E.32.1 Detailed Description . . . . .	97
E.32.2 Constructor & Destructor Documentation . . . . .	97
E.32.2.1 StrategyError . . . . .	97
E.32.2.2 StrategyError . . . . .	97



# 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 established 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.<sup>1</sup>

### 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<sup>2</sup> 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<sup>3</sup> and iris IREXIII<sup>4</sup>) 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.

---

<sup>1</sup><http://www.nist.gov/itl/iad/ig/fpvte03.cfm>

<sup>2</sup><http://www.nist.gov/itl/iad/ig/fpvte2012.cfm>

<sup>3</sup><http://www.nist.gov/itl/iad/ig/mbe.cfm>

<sup>4</sup><http://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).<sup>5</sup>

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.

---

<sup>5</sup><http://www.nist.gov/itl/iad/ig/pftii.cfm>

Enrollment Sets				
Set Name	Finger(s)	Data Type	Enrolled Sizes to be Tested	Subject
E1 (Class A)	1f Right Index	Plain Capture	5,000 10,000 100,000	
E2 (Class A)	1f Left Index	Plain Capture	5,000 10,000 100,000	
E3 (Class A)	2f Right and Left Index	Plain Capture	10,000 100,000 500,000 1,600,000	
E4 (Class B)	10f Identification Flats (4-4-2)	Plain Capture	500,000 1,600,000 3,000,000	
E5 (Class C)	10f	Rolled	500,000 1,600,000 3,000,000 5,000,000	
E6 (Class C)	10f (4-4-1-1)	Plain Capture	500,000 1,600,000 3,000,000 5,000,000	

Search Sets		
Set Name	Finger(s)	Data Type
S1 (Class A)	1f Right Index	Plain Capture
S2 (Class A)	1f Left Index	Plain Capture
S3 (Class A)	2f Right and Left Index	Plain Capture
S4 (Class B)	4f Right Slap ID Flat	Plain Capture
S5 (Class B)	4f Left Slap ID Flat	Plain Capture
S6 (Class B)	8f Right and Left Slap ID Flat	Plain Capture
S7 (Class B)	10f ID Flats (4-4-2)	Plain Capture
S8 (Class C)	10f	Rolled
S9 (Class C)	10f (4-4-1-1)	Plain Capture

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	
Class	Search vs Enrollment Sets
A	S1 x E1 S2 x E2 S3 x E3
B	S4 x E4 S5 x E4 S6 x E4 S7 x E4
C	S8 x E5 S9 x E5 S9 x E6

## 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:

- machine generated (i.e. scripted),
- provided to participants with identification of their algorithm,
- include template size, timing, and performance accuracy results (ie. 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 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.

## 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:

- /usr/lib64/libstdc++.so.6.0.13 (GLIBCXX 3.4.13)

- /lib64/libc.so.6 -> libc-2.12.so (GLIBC 2.12)
- /lib64/libm.so.6 -> libm-2.12.so

and a typical link command would be:

```
g++ -I. -Wall -m64 -o fpvtel2test fpvtel2test.cpp -L. -lfpvtel2_Participant_A
```

## 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. 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.

Search Timing by Class		
Class	Search vs Enrollment Set	Time per Search
A	S1 x E1	500s
	S2 x E2	
	S3 x E3	
B	S4 x E4	90s
	S5 x E4	
	S6 x E4	
	S7 x E4	
C	S8 x E5	90s
	S9 x E5	
	S9 x E6	

## 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.

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 `FingerImages`. 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`.

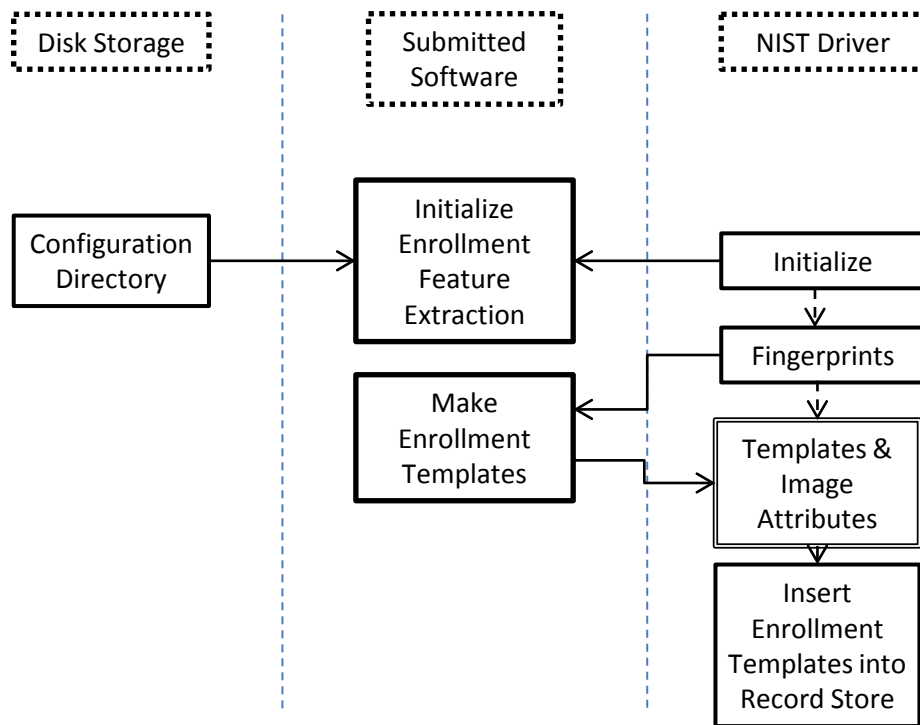


Figure 3.1: Flowchart for enrollment feature template extraction process

### 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.

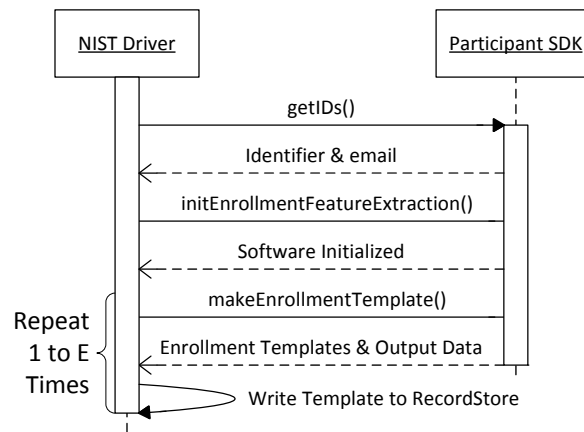


Figure 3.2: Calling sequence for enrollment feature template extraction process

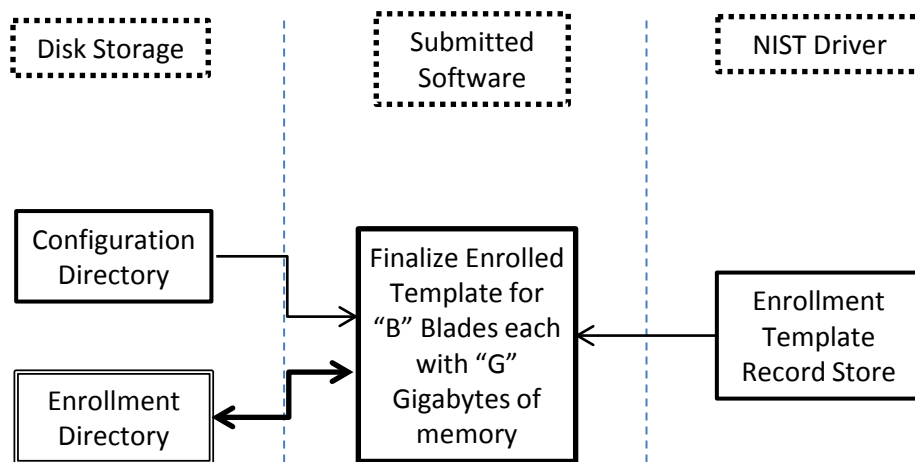


Figure 3.3: Flowchart of enrollment template finalization process

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.

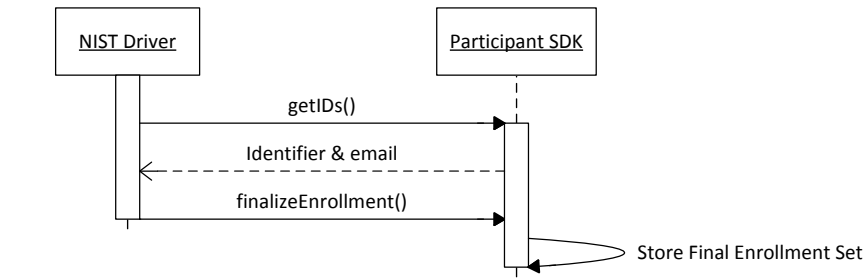


Figure 3.4: Calling sequence for enrollment template finalization process

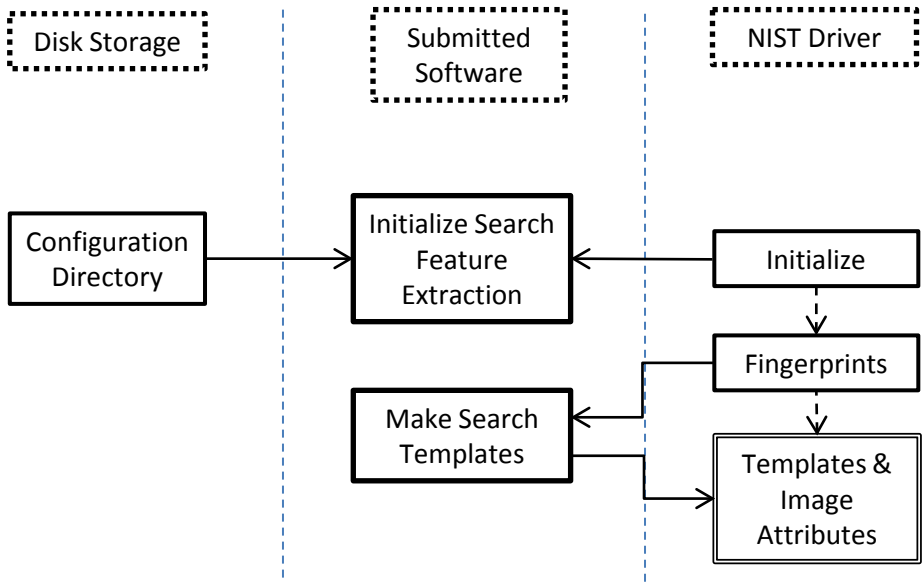


Figure 3.5: Flowchart of search feature template extraction process

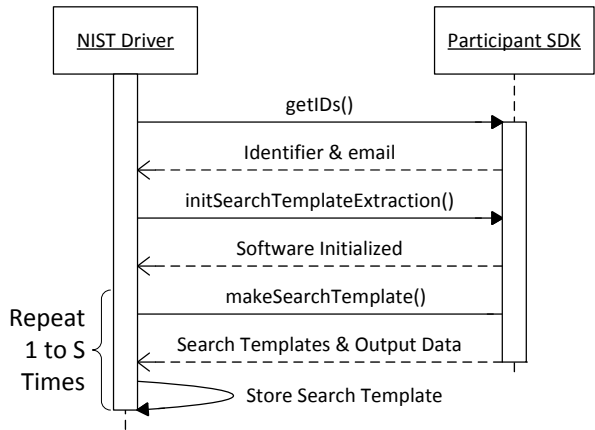


Figure 3.6: Calling sequence for search feature template extraction process

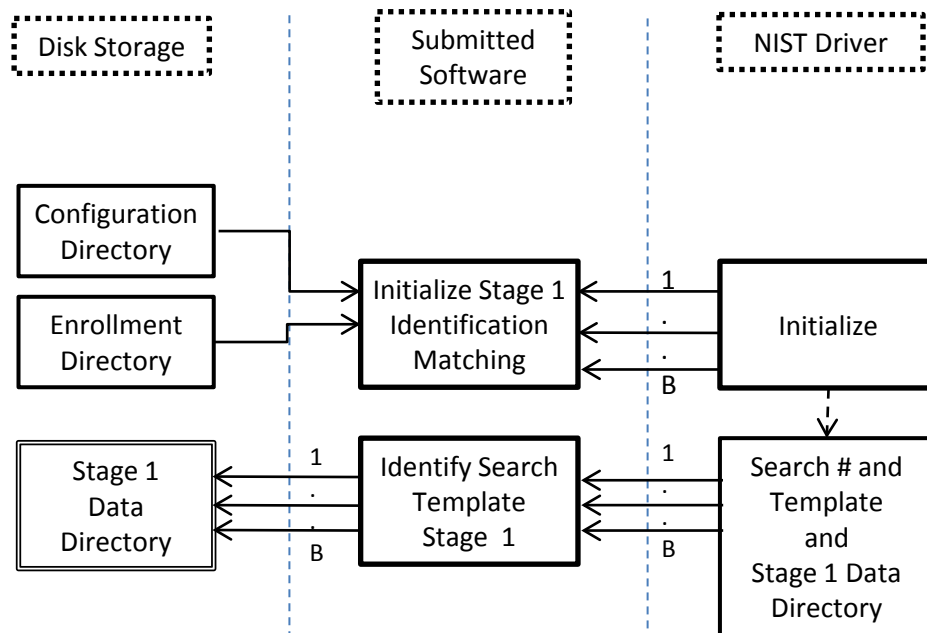


Figure 3.7: Flowchart of first stage of identification process

### 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).

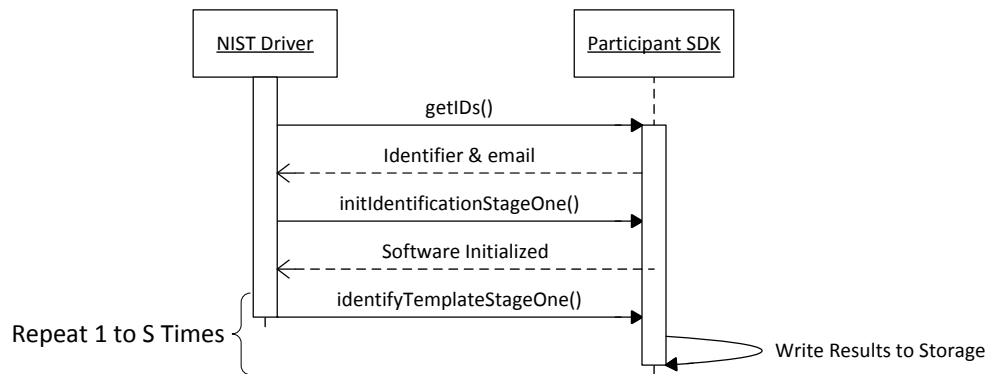


Figure 3.8: Calling sequence for first stage of identification process

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:

#### Listing 3.1: SDKInterface Subclass Declaration

```

1 #include <fpvte2012.h>
2
3 namespace FpVTE2012 {
4   class NullSDK : public FpVTE2012::SDKInterface {
5   public:
6
7       NullSDK();
8       ~NullSDK();
9
10      void getIDs(

```



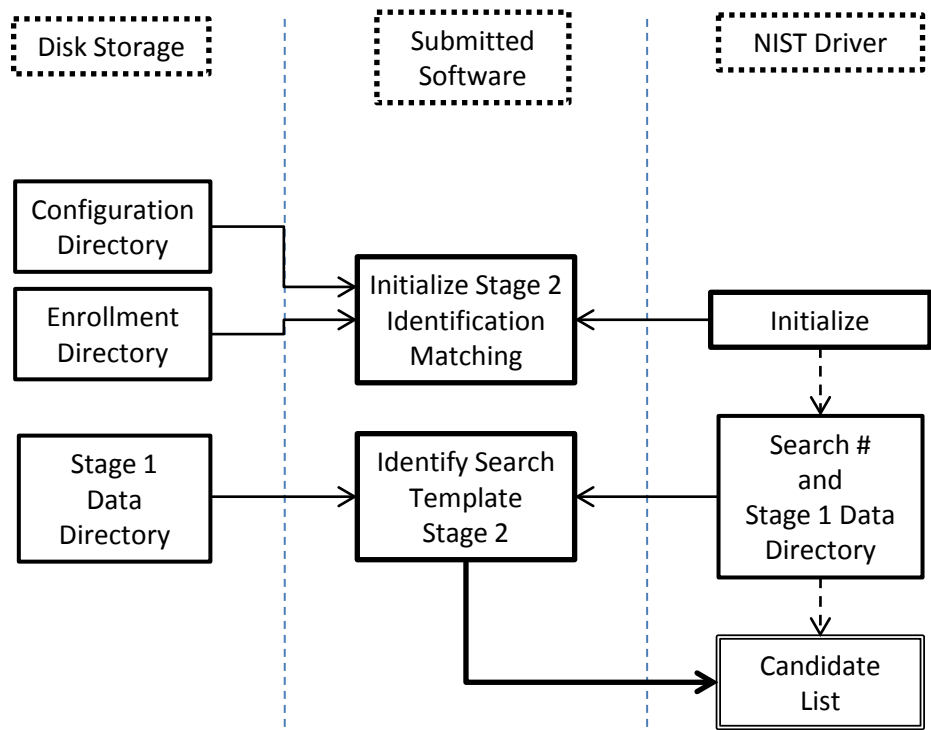


Figure 3.9: Flowchart of second stage of identification process

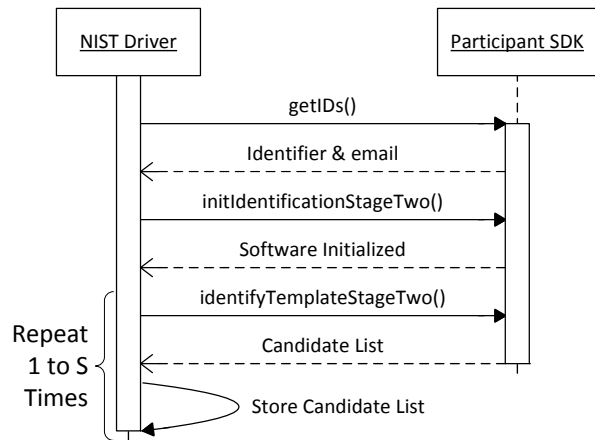


Figure 3.10: Calling sequence for second stage of identification process

```

11         string &nist_assigned_identifier,
12         string &email_address);
13 }

```

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:

**Listing 3.2: SDKInterface Subclass Definition**

```

1 #include <fpvteNullSDK.h>
2
3 using namespace BiometricEvaluation;
4 using namespace FPVTE2012;
5
6 NullSDK::NullSDK () { }
7
8 NullSDK::~NullSDK() { }
9
10 void
11 NullSDK::getIDs(
12     string &nist_assigned_identifier,
13     string &email_address)
14 {
15     nist_assigned_identifier = "0x12345678";
16     email_address = "foo@example.com";
17 }
18
19 SDKIptr
20 SDKInterface::getSDK()
21 {
22     NullSDK *p = new NullSDK();
23     SDKIptr ap(p);
24     return (ap);
25 }

```

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_[1 or 2]_sequence.[so or a]`

**NIST\_Assigned\_ID** The implementation identifier assigned by the NIST test liaison.

**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/evaluation/6.2x86\\_64binDVD1.iso](http://nigos.nist.gov:8080/evaluation/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 to 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.

#### 4.6.4 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

[1] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, special edition, 2000. [16](#)

# Appendix A

## Namespace Index

### A.1 Namespace List

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

<a href="#">BiometricEvaluation::Error</a>	
Exceptions, and other error handling . . . . .	30
<a href="#">BiometricEvaluation::Finger</a>	
Biometric information relating to finger images and derived information . . . . .	31
<a href="#">BiometricEvaluation::Image</a>	
Basic information relating to images . . . . .	32
<a href="#">BiometricEvaluation::IO</a>	
Input/Output functionality . . . . .	33
<a href="#">BiometricEvaluation::IO::Utility</a>	34
<a href="#">BiometricEvaluation::Text</a>	
Text processing for string objects . . . . .	36
<a href="#">FPVTE2012</a>	
Contains all the structures and functions used by the API . . . . .	36

# Appendix B

## Class Index

### B.1 Class Hierarchy

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

BiometricEvaluation::Memory::AutoArray< T > . . . . .	39
FPVTE2012::Candidate . . . . .	45
BiometricEvaluation::Image::CompressionAlgorithm . . . . .	45
BiometricEvaluation::Image::Coordinate . . . . .	47
BiometricEvaluation::Error::Exception . . . . .	49
BiometricEvaluation::Error::ConversionError . . . . .	46
BiometricEvaluation::Error::DataError . . . . .	48
BiometricEvaluation::Error::FileError . . . . .	51
BiometricEvaluation::Error::MemoryError . . . . .	64
BiometricEvaluation::Error::NotImplemented . . . . .	64
BiometricEvaluation::Error::ObjectDoesNotExist . . . . .	65
BiometricEvaluation::Error::ObjectExists . . . . .	66
BiometricEvaluation::Error::ObjectIsClosed . . . . .	67
BiometricEvaluation::Error::ObjectIsOpen . . . . .	68
BiometricEvaluation::Error::ParameterError . . . . .	69
BiometricEvaluation::Error::StrategyError . . . . .	97
FPVTE2012::FingerImage . . . . .	56
BiometricEvaluation::Finger::FingerImageCode . . . . .	57
FPVTE2012::FingerOutput . . . . .	57
BiometricEvaluation::Image::Image . . . . .	58
BiometricEvaluation::Image::Raw . . . . .	76
BiometricEvaluation::Finger::Impression . . . . .	63
BiometricEvaluation::Finger::PatternClassification . . . . .	70
BiometricEvaluation::Finger::Position . . . . .	71
BiometricEvaluation::IO::Properties . . . . .	71
BiometricEvaluation::IO::RecordStore . . . . .	77
BiometricEvaluation::IO::FileRecordStore . . . . .	51
BiometricEvaluation::Image::Resolution . . . . .	88
FPVTE2012::ReturnStatus . . . . .	89
FPVTE2012::SDKInterface . . . . .	90
BiometricEvaluation::Image::Size . . . . .	95
FPVTE2012::StatusCode . . . . .	96

# Appendix C

## Class Index

### C.1 Class List

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

<a href="#">BiometricEvaluation::Memory::AutoArray&lt; T &gt;</a>	
A C-style array wrapped in the facade of a C++ STL container . . . . .	39
<a href="#">FPVTE2012::Candidate</a>	
Object defining format of candidate reporting . . . . .	45
<a href="#">BiometricEvaluation::Image::CompressionAlgorithm</a>	
Image compression algorithms . . . . .	45
<a href="#">BiometricEvaluation::Error::ConversionError</a>	
Error when converting one object into another, a property value from string to int, for example . . . . .	46
<a href="#">BiometricEvaluation::Image::Coordinate</a>	
A structure to contain a two-dimensional coordinate without a specified origin . . . . .	47
<a href="#">BiometricEvaluation::Error::DataError</a>	
Error when reading data from an external source . . . . .	48
<a href="#">BiometricEvaluation::Error::Exception</a>	
The parent class of all BiometricEvaluation exceptions . . . . .	49
<a href="#">BiometricEvaluation::Error::FileError</a>	
File error when opening, reading, writing, etc . . . . .	51
<a href="#">BiometricEvaluation::IO::FileRecordStore</a>	
<a href="#">FPVTE2012::FingerImage</a>	
Object containing the fingerprint image and corresponding attributes . . . . .	56
<a href="#">BiometricEvaluation::Finger::FingerImageCode</a>	
<a href="#">FPVTE2012::FingerOutput</a>	
Object for output data related to an extracted template . . . . .	57
<a href="#">BiometricEvaluation::Image::Image</a>	
Represent attributes common to all images . . . . .	58
<a href="#">BiometricEvaluation::Finger::Impression</a>	
Finger and palm impression types . . . . .	63
<a href="#">BiometricEvaluation::Error::MemoryError</a>	
An error occurred when allocating an object . . . . .	64
<a href="#">BiometricEvaluation::Error::NotImplemented</a>	
A <a href="#">NotImplemented</a> object is thrown when the underlying implementation of this interface has not or could not be created . . . . .	64
<a href="#">BiometricEvaluation::Error::ObjectDoesNotExist</a>	
The named object does not exist . . . . .	65

<a href="#">BiometricEvaluation::Error::ObjectExists</a>	
The named object exists and will not be replaced . . . . .	66
<a href="#">BiometricEvaluation::Error::ObjectIsClosed</a>	
The object is closed . . . . .	67
<a href="#">BiometricEvaluation::Error::ObjectIsOpen</a>	
The object is already opened . . . . .	68
<a href="#">BiometricEvaluation::Error::ParameterError</a>	
An invalid parameter was passed to a constructor or method . . . . .	69
<a href="#">BiometricEvaluation::Finger::PatternClassification</a>	
Pattern classification codes . . . . .	70
<a href="#">BiometricEvaluation::Finger::Position</a>	
Finger position codes . . . . .	71
<a href="#">BiometricEvaluation::IO::Properties</a>	
A <a href="#">Properties</a> class is used to maintain key/value pairs of strings, with each property matched to one value . . . . .	71
<a href="#">BiometricEvaluation::Image::Raw</a>	
An image with no encoding or compression . . . . .	76
<a href="#">BiometricEvaluation::IO::RecordStore</a>	
A class to represent a data storage mechanism . . . . .	77
<a href="#">BiometricEvaluation::Image::Resolution</a>	
A structure to represent the resolution of an image . . . . .	88
<a href="#">FPVTE2012::ReturnStatus</a>	
A structure to contain information about a failure by the software under test . . . . .	89
<a href="#">FPVTE2012::SDKInterface</a>	
. . . . .	90
<a href="#">BiometricEvaluation::Image::Size</a>	
A structure to represent the size of an image, in pixels . . . . .	95
<a href="#">FPVTE2012::StatusCode</a>	
. . . . .	96
<a href="#">BiometricEvaluation::Error::StrategyError</a>	
A <a href="#">StrategyError</a> object is thrown when the underlying implementation of this interface encounters an error . . . . .	97

# 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	<i>stream</i>	Stream on which to append formatted CoordinateSet information.
in	<i>coordinates</i>	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	<i>p1</i>	First point.
in	<i>p2</i>	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](#)

A [Properties](#) class is used to maintain key/value pairs of strings, with each property matched to one value.

- class [RecordStore](#)

A class to represent a data storage mechanism.

## Typedefs

- typedef map< string, string > **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 Variable Documentation

#### D.4.2.1 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.4.2.2 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)
- uint64\_t **getFileSize** (const string &pathname) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- bool **fileExists** (const string &pathname) throw (Error::StrategyError)
- bool **validateRootName** (const string &name)
- bool **constructAndCheckPath** (const string &name, const string &parentDir, string &fullPath)

### 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

in	<i>directory</i>	The name of the directory to be removed, without a preceding path.
in	<i>prefix</i>	The path leading to the directory.

### Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The named directory does not exist.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system, or the directoy name or prefix is malformed.

#### D.5.2.2 uint64\_t BiometricEvaluation::IO::Utility::getFileSize ( const string & *pathname* ) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Get the size of a file.

### Parameters

in	<i>pathname</i>	The name of the file to be sized; can be a complete path.
----	-----------------	---

### Returns

The file size.

### Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The named directory does not exist.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system, or pathname is malformed.

#### D.5.2.3 bool BiometricEvaluation::IO::Utility::fileExists ( const string & *pathname* ) throw (Error::StrategyError)

Indicate whether a file exists.

### Parameters

in	<i>pathname</i>	The name of the file to be checked; can be a complete path.
----	-----------------	---

### Returns

true if the file exists, false otherwise.

### Exceptions

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system, or pathname is malformed.
---	---

#### D.5.2.4 bool BiometricEvaluation::IO::Utility::validateRootName ( const string & *name* )

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

in	<i>name</i>	The proposed name for the entity.
----	-------------	-----------------------------------

### Returns

true if the name is acceptable, false otherwise.

#### D.5.2.5 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	<i>name</i>	The proposed name for the entity; cannot be a pathname.
in	<i>parentDir</i>	The name of the directory to contain the entity.
out	<i>fullPath</i>	The complete path to the new entity, when when true is returned; ambiguous when false is returned.

### Returns

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

## 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.*

#### 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.7 FPVTE2012 Namespace Reference

Contains all the structures and functions used by the API.

### 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 [Memory::AutoArray](#) < [FingerImage](#) > [FingerImageSet](#)
- typedef struct [FingerOutput](#) **FingerOutput**
- typedef [Memory::AutoArray](#) < [FingerOutput](#) > [FingerOutputSet](#)
- typedef struct [Candidate](#) **Candidate**
- typedef [Memory::AutoArray](#) < [Candidate](#) > [CandidateSet](#)
- typedef struct [ReturnStatus](#) **ReturnStatus**
- typedef [tr1::shared\\_ptr](#) < [SDKInterface](#) > **SDKIptr**

## Functions

- [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 [Memory::AutoArray](#)<[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 [Memory::AutoArray](#)<[FingerOutput](#)> [FPVTE2012::FingerOutputSet](#)

Object representing a set of Finger output records. The number of records output should be equal to the number of records input to the process.

#### D.7.2.3 typedef [Memory::AutoArray](#)<[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.7.3 Function Documentation

#### D.7.3.1 [std::ostream](#)& [FPVTE2012::operator<<](#) ( [std::ostream](#) & s, const [FPVTE2012::StatusCode::Kind](#) & sc )

Output stream operator for a [StatusCode](#) object.

### D.7.3.2 `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)  
*Subscript into the [AutoArray](#) with checked access.*
- [const\\_reference at](#) (ptrdiff\_t index) const throw (out\_of\_range)  
*Subscript 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::StrategyError)
- 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](#) ()
- Construct an [AutoArray](#).*
- [AutoArray](#) ([size\\_type](#) size)
- Construct an [AutoArray](#).*
- [AutoArray](#) (const [AutoArray](#) &copy)
- Construct an [AutoArray](#).*
- [AutoArray](#) & [operator=](#) (const [AutoArray](#) &other)
- 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 ( )`

Construct an [AutoArray](#).

The [AutoArray](#) will be of size 0.

**E.1.3.2** `template<class T> BiometricEvaluation::Memory::AutoArray< T >::AutoArray ( size_type size )`

Construct an [AutoArray](#).

**Parameters**

<i>in</i>	<i>size</i>	The number of elements this <a href="#">AutoArray</a> should initially hold.
-----------	-------------	--

**E.1.3.3** `template<class T> BiometricEvaluation::Memory::AutoArray< T >::AutoArray ( const AutoArray< T > & copy )`

Construct an [AutoArray](#).

**Parameters**

<i>in</i>	<i>copy</i>	An <a href="#">AutoArray</a> whose contents will be deep copied into the new <a href="#">AutoArray</a> .
-----------	-------------	--

**E.1.3.4** `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< 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

<code>in</code>	<code>index</code>	Subscript into underlying storage.
-----------------	--------------------	------------------------------------

#### Returns

Reference to the element at the specified index.

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

Const subscripting operator overload with unchecked access.

#### Parameters

<code>in</code>	<code>index</code>	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

<code>in</code>	<code>index</code>	Subscript into underlying storage.
-----------------	--------------------	------------------------------------

#### Returns

Reference to the element at the specified index.

#### Exceptions

<code>out_of_range</code>	Specified index is outside the bounds of this <a href="#">AutoArray</a> .
---------------------------	---

**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

<i>index</i>	Subscript into underlying storage.
--------------	------------------------------------

**Returns**

Const reference to the element at the specified index.

**Exceptions**

<i>out_of_range</i>	Specified index is outside the bounds of this <a href="#">AutoArray</a> .
---------------------	---

#### 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::StrategyError)`

Change the number of accessible elements.

#### Parameters

in	<i>new_size</i>	The number of elements the <code>AutoArray</code> should have allocated.
in	<i>free</i>	Whether or not excess memory should be freed if the new size is smaller than the current size.

#### Exceptions

<code>Error::StrategyError</code>	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

in	<i>buffer</i>	An allocated buffer whose contents will be deep-copied into this object. Only <code>size()</code> bytes will be copied.
----	---------------	---

**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

in	<i>buffer</i>	An allocated buffer whose contents will be deep-copied into this object.
in	<i>size</i>	The number of bytes from buffer that will be deep-copied.

**E.1.4.15** `template<class T> BiometricEvaluation::Memory::AutoArray< T> & BiometricEvaluation::Memory::AutoArray< T>::operator= ( const AutoArray< T> & other )`

Assignment operator overload performing a deep copy.

#### Parameters

<i>in</i>	<i>other</i>	<a href="#">AutoArray</a> to be copied.
-----------	--------------	---

#### Returns

Reference to a new [AutoArray](#) object, the lvalue [AutoArray](#).

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

- `uint32_t` **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

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

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:

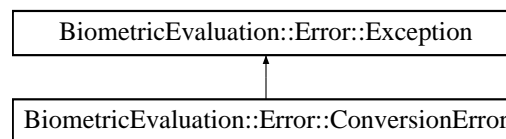
- be\_image.h

## 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:



## 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

in	<a href="#">x</a>	X-coordinate
in	<a href="#">y</a>	Y-coordinate
in	<a href="#">xDistance</a>	X-coordinate distance from origin
in	<a href="#">yDistance</a>	Y-coordinate distance from origin

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:

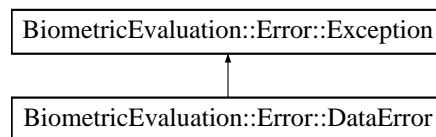
- be\_image.h

## 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:



### 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:

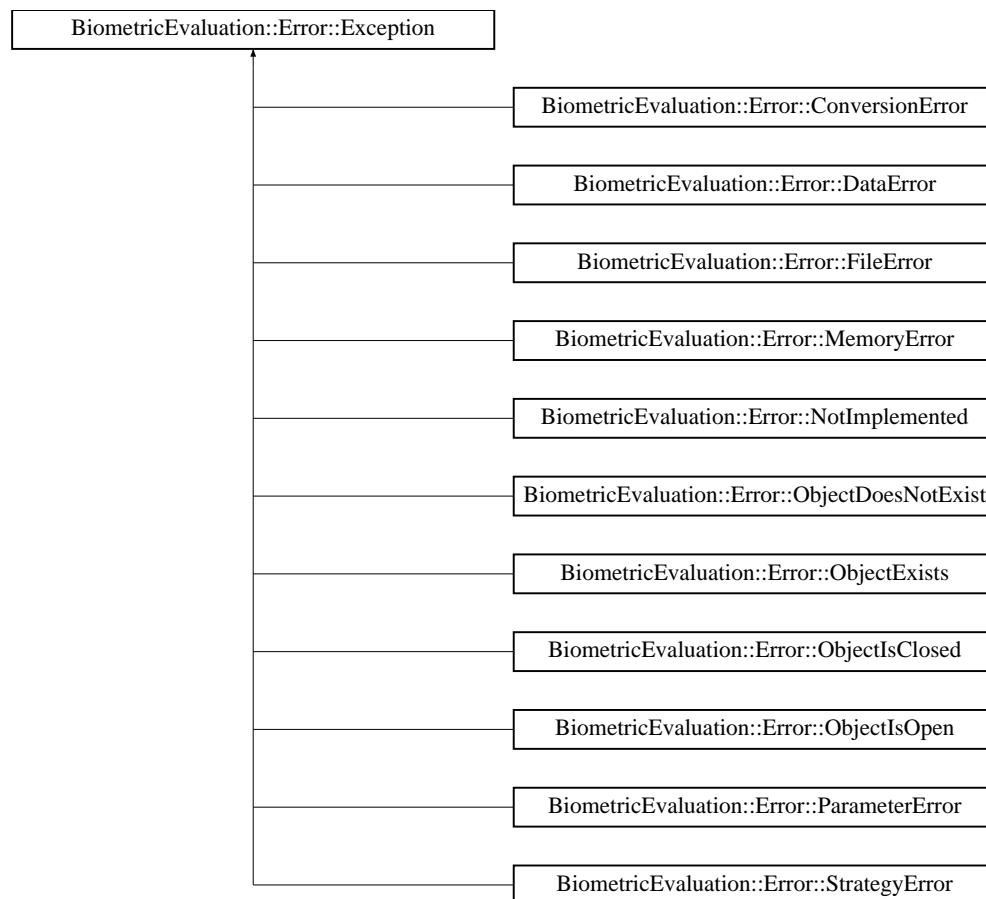
- `be_error_exception.h`

## 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:



## 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.7.2.2 BiometricEvaluation::Error::Exception::Exception ( string info )

Construct an [Exception](#) object with an information string.

##### Parameters

<i>in</i>	<i>info</i>	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:

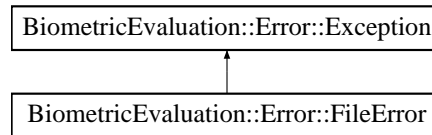
- be\_error\_exception.h

## 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:



### Public Member Functions

- [FileError](#) ()
- [FileError](#) (string info)

#### E.8.1 Detailed Description

File error when opening, reading, writing, etc.

#### 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.

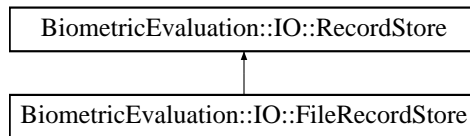
The documentation for this class was generated from the following file:

- be\_error\_exception.h

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

```
#include <be_io_filerecstore.h>
```

Inheritance diagram for BiometricEvaluation::IO::FileRecordStore:



## 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::ObjectDoesNotExist, 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, 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)
- 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

### E.9.1 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.

### E.9.2 Constructor & Destructor Documentation

- E.9.2.1** [BiometricEvaluation::IO::FileRecordStore::FileRecordStore](#) ( const string & *name*, const string & *description*, const string & *parentDir* ) throw (Error::ObjectExists, Error::StrategyError)

Create a new [FileRecordStore](#), read/write mode.

#### Parameters

in	<i>name</i>	The name of the store.
in	<i>description</i>	The store's description.
in	<i>parentDir</i>	The directory where the store is to be created.

## Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	The store already exists.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when accessing the underlying file system.

### E.9.2.2 BiometricEvaluation::IO::FileRecordStore::FileRecordStore ( const string & *name*, const string & *parentDir*, uint8\_t *mode* = IO::READWRITE ) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing [FileRecordStore](#).

## Parameters

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

## Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The store does not exist.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when accessing the underlying file system.

## 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

<a href="#"><i>Error::StrategyError</i></a>	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	<i>key</i>	The key of the record to be inserted.
in	<i>data</i>	The data for the record.
in	<i>size</i>	The size, in bytes, of the record.

**Exceptions**

<a href="#"><i>Error::ObjectExists</i></a>	A record with the given key is already present.
<a href="#"><i>Error::StrategyError</i></a>	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**

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	A record for the key does not exist.
<a href="#"><i>Error::StrategyError</i></a>	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**

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	A record for the key does not exist.
<a href="#"><i>Error::StrategyError</i></a>	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**

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	A record for the key does not exist.
<a href="#"><i>Error::StrategyError</i></a>	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**

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	A record for the key does not exist.
<a href="#"><i>Error::StrategyError</i></a>	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**

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	A record for the key does not exist.
<a href="#"><i>Error::StrategyError</i></a>	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 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	<i>key</i>	The key of the currently sequenced record.
in	<i>data</i>	Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.
in	<i>cursor</i>	The location within the sequence of the key/data pair to return.

**Returns**

The length of the record currently in sequence.

**Exceptions**

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	A record for the key does not exist.
<a href="#"><i>Error::StrategyError</i></a>	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	<i>key</i>	The key of the record which will be returned by the first subsequent call to <a href="#">sequence()</a> .
----	------------	---

**Exceptions**

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	A record for the key does not exist.
<a href="#"><i>Error::StrategyError</i></a>	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	<i>name</i>	The new name for the <a href="#">RecordStore</a> .
----	-------------	--

**Exceptions**

<a href="#"><i>Error::StrategyError</i></a>	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

<i>fgp</i>	Fingerprint position.
<i>imp</i>	Impression type for the fingerprint.
<i>raw_image</i>	Input image data.
<i>nfiq</i>	Image nfiq value with range from 1 (best) ... 5 (worst).

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

- `uint16_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

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

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

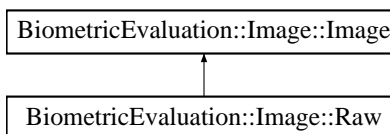
- fpvte2012.h

## E.13 BiometricEvaluation::Image::Image Class Reference

Represent attributes common to all images.

```
#include <be_image_image.h>
```

Inheritance diagram for BiometricEvaluation::Image::Image:



## 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

<code>in</code>	<code><i>data</i></code>	The image data.
<code>in</code>	<code><i>size</i></code>	The size of the image data, in bytes.
<code>in</code>	<code><i>dimensions</i></code>	The width and height of the image in pixels.
<code>in</code>	<code><i>depth</i></code>	The image depth, in bits-per-pixel.
<code>in</code>	<code><i>resolution</i></code>	The resolution of the image
<code>in</code>	<code><i>compression</i></code>	The <a href="#">CompressionAlgorithm</a> of data.

### Exceptions

<a href="#">Error::StrategyError</a>	Error manipulating data.
<a href="#">Error::StrategyError</a>	Error while creating <a href="#">Image</a> .

**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

<code>in</code>	<code><i>data</i></code>	The image data.
<code>in</code>	<code><i>size</i></code>	The size of the image data, in bytes.
<code>in</code>	<code><i>compression</i></code>	The <a href="#">CompressionAlgorithm</a> of data.

### Exceptions

<a href="#">Error::DataError</a>	Error manipulating data.
<a href="#">Error::StrategyError</a>	Error while creating <a href="#">Image</a> .

## 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**

<a href="#">Error::DataError</a>	<a href="#">Error</a> decompressing image data.
----------------------------------	---

Implemented in [BiometricEvaluation::Image::Raw](#).

**E.13.3.5 virtual Memory::AutoArray<uint8\_t>** BiometricEvaluation::Image::Image::getRawGrayscaleData ( uint8\_t *depth* = 8 ) const throw (Error::DataError, Error::ParameterError) [pure virtual]

Accessor for decompressed data in grayscale.

**Parameters**

<i>depth</i>	The desired bit depth of the resulting raw image. This value may either be 8 or 1.
--------------	--

**Returns**

[Raw](#) image buffer.

**Exceptions**

<a href="#">Error::DataError</a>	<a href="#">Error</a> decompressing image data.
<a href="#">Error::ParameterError</a>	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::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::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::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

<i>color</i>	Value for color in original colorspace.
<i>maxColorValue</i>	Maximum value for colors in original colorspace.
<i>depth</i>	Desired bit-depth of the new colorspace.

#### 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	<i>resolution</i>	<a href="#">Resolution</a> 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	<i>dimensions</i>	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	<i>depth</i>	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

**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, LiveScanNonOpticalContactlessPlain, LiveScanNonOpticalContactlessRolled, Other, Unknown }

**E.14.1 Detailed Description**

[Finger](#) and palm impression types.

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

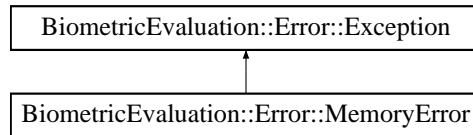
- be\_finger.h

## 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:



### 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::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

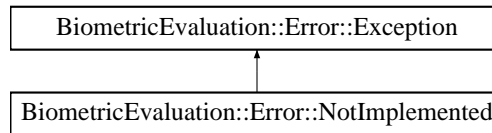
## E.16 BiometricEvaluation::Error::NotImplemented Class Reference

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:



## 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 info )

Construct a [NotImplemented](#) object with an information string appended to the default information string.

#### Returns

The [NotImplemented](#) object.

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

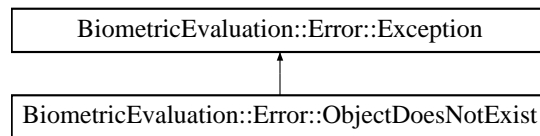
- be\_error\_exception.h

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

The named object does not exist.

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

Inheritance diagram for BiometricEvaluation::Error::ObjectDoesNotExist:



## Public Member Functions

- [ObjectDoesNotExist](#) ()
- [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 [ObjectDoesNotExist](#) object with the default information string.

#### Returns

The [ObjectDoesNotExist](#) object.

#### E.17.2.2 BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( string info )

Construct a [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:

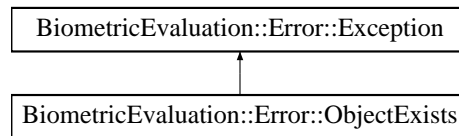
- be\_error\_exception.h

## 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:



## 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:

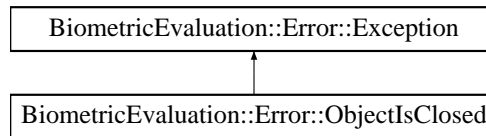
- be\_error\_exception.h

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

The object is closed.

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

Inheritance diagram for 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:

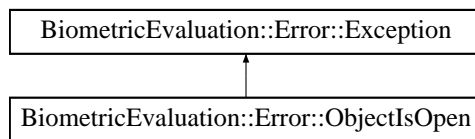
- be\_error\_exception.h

## E.20 BiometricEvaluation::Error::ObjectIsOpen Class Reference

The object is already opened.

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

Inheritance diagram for BiometricEvaluation::Error::ObjectIsOpen:



## 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:

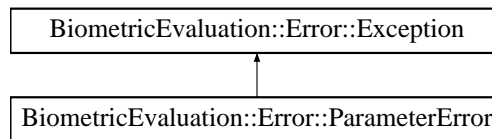
- `be_error_exception.h`

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

An invalid parameter was passed to a constructor or method.

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

Inheritance diagram for BiometricEvaluation::Error::ParameterError:



## Public Member Functions

- [ParameterError](#) ()
- [ParameterError](#) (string info)

### E.21.1 Detailed Description

An invalid parameter was passed to a constructor or method.

### E.21.2 Constructor & Destructor Documentation

#### E.21.2.1 BiometricEvaluation::Error::ParameterError::ParameterError ( )

Construct a [ParameterError](#) object with the default information string.

#### Returns

The [ParameterError](#) object.

#### E.21.2.2 BiometricEvaluation::Error::ParameterError::ParameterError ( string info )

Construct a [ParameterError](#) object with an information string appended to the default information string.

#### Returns

The [ParameterError](#) object.

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

- 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

A [Properties](#) class is used to maintain key/value pairs of strings, with each property matched to one value.

```
#include <be_io_properties.h>
```

### Public Types

- typedef `PropertiesMap::const_iterator` **Properties\_iter**

### Public Member Functions

- [Properties](#) (const string &filename, uint8\_t mode=[IO::READWRITE](#)) throw (Error::StrategyError, Error::FileError)
- [Properties](#) (const uint8\_t \*buffer, const size\_t size) throw (Error::StrategyError)
- void [setProperty](#) (const string &property, const string &value) throw (Error::StrategyError)
- void [setPropertyFromInteger](#) (const string &property, int64\_t value) throw (Error::StrategyError)
- void [setPropertyFromDouble](#) (const string &property, double value) throw (Error::StrategyError)
- void [removeProperty](#) (const string &property) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- string [getProperty](#) (const string &property) throw (Error::ObjectDoesNotExist)
- int64\_t [getPropertyAsInteger](#) (const string &property) throw (Error::ObjectDoesNotExist, Error::ConversionError)

- double [getPropertyAsDouble](#) (const string &property) throw (Error::ObjectDoesNotExist)
- void [sync](#) () throw (Error::FileError, Error::StrategyError)
- void [changeName](#) (const string &filename) throw (Error::StrategyError)

### E.24.1 Detailed Description

A [Properties](#) class is used to maintain key/value pairs of strings, with each property matched to one value.

The properties are read from a file that is specified in the constructor, and will be created if it does not exist.

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 a the call

```
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.24.2 Constructor & Destructor Documentation

#### E.24.2.1 BiometricEvaluation::IO::Properties::Properties ( const string & filename, uint8\_t mode = IO::READWRITE ) throw (Error::StrategyError, Error::FileError)

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

in	<i>filename</i>	The name of the file to store the properties. This can be the empty string, meaning the properties are to be stored in memory only.
in	<i>mode</i>	The read/write mode of the object.

#### Returns

An object representing the properties set.

#### Exceptions

<a href="#">Error::StrategyError</a>	A line in the properties file is malformed.
<a href="#">Error::FileError</a>	An error occurred when using the underlying storage system.



### E.24.2.2 BiometricEvaluation::IO::Properties::Properties ( `const uint8_t * buffer, const size_t size` ) throw (Error::StrategyError)

Construct a new [Properties](#) object from the contents of a buffer.

#### Parameters

<code>in</code>	<code><i>buffer</i></code>	A buffer that contains the contents of a Property file.
<code>in</code>	<code><i>size</i></code>	The size of buffer.

#### Returns

An object representing the properties set.

#### Exceptions

<a href="#">Error::StrategyError</a>	A line in the properties file is malformed.
--------------------------------------	---

## E.24.3 Member Function Documentation

### E.24.3.1 void BiometricEvaluation::IO::Properties::setProperty ( `const string & property, const string & value` ) throw (Error::StrategyError)

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

<code>in</code>	<code><i>property</i></code>	The name of the property to set.
<code>in</code>	<code><i>value</i></code>	The value associated with the property.

#### Exceptions

<a href="#">Error::StrategyError</a>	The <a href="#">Properties</a> object is read-only.
--------------------------------------	---

### E.24.3.2 void BiometricEvaluation::IO::Properties::setPropertyFromInteger ( `const string & property, int64_t value` ) throw (Error::StrategyError)

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

<code>in</code>	<code><i>property</i></code>	The name of the property to set.
<code>in</code>	<code><i>value</i></code>	The value associated with the property.

#### Exceptions

<a href="#">Error::StrategyError</a>	The <a href="#">Properties</a> object is read-only.
--------------------------------------	---

**E.24.3.3** void BiometricEvaluation::IO::Properties::setPropertyFromDouble ( const string & *property*, double *value* ) throw (Error::StrategyError)

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	<i>property</i>	The name of the property to set.
in	<i>value</i>	The value associated with the property.

#### Exceptions

<a href="#">Error::StrategyError</a>	The <a href="#">Properties</a> object is read-only.
--------------------------------------	---

**E.24.3.4** void BiometricEvaluation::IO::Properties::removeProperty ( const string & *property* ) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Remove a property.

#### Parameters

in	<i>property</i>	The name of the property to set.
----	-----------------	----------------------------------

#### Exceptions

<a href="#">Error::ObjectDoesNotExist</a>	The named property does not exist.
<a href="#">Error::StrategyError</a>	The <a href="#">Properties</a> object is read-only.

**E.24.3.5** string BiometricEvaluation::IO::Properties::getProperty ( const string & *property* ) throw (Error::ObjectDoesNotExist)

Retrieve a property value as a string object.

#### Parameters

in	<i>property</i>	The name of the property to get.
----	-----------------	----------------------------------

#### Exceptions

<a href="#">Error::ObjectDoesNotExist</a>	The named property does not exist.
---	------------------------------------

**E.24.3.6** int64\_t BiometricEvaluation::IO::Properties::getPropertyAsInteger ( const string & *property* ) throw (Error::ObjectDoesNotExist, Error::ConversionError)

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	<i>property</i>	The name of the property to get.
----	-----------------	----------------------------------

### Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The named property does not exist.
<a href="#"><i>Error::ConversionError</i></a>	The property value cannot be converted, usually due to non-numeric characters in the string.

#### E.24.3.7 double BiometricEvaluation::IO::Properties::getPropertyAsDouble ( const string & *property* ) throw (Error::ObjectDoesNotExist)

Retrieve a property value as a double value.

### Parameters

in	<i>property</i>	The name of the property to get.
----	-----------------	----------------------------------

### Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The named property does not exist.
--	------------------------------------

#### E.24.3.8 void BiometricEvaluation::IO::Properties::sync ( ) throw (Error::FileError, Error::StrategyError)

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

### Exceptions

<a href="#"><i>Error::FileError</i></a>	An error occurred when using the underlying storage system.
<a href="#"><i>Error::StrategyError</i></a>	The object was constructed with NULL as the file name, or is read-only.

#### E.24.3.9 void BiometricEvaluation::IO::Properties::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

in	<i>filename</i>	The name of the properties file.
----	-----------------	----------------------------------

### Exceptions

<a href="#"><i>Error::StrategyError</i></a>	The object is read-only.
---	--------------------------

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

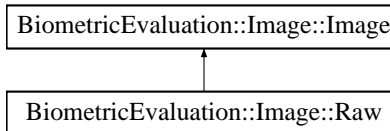
- be\_io\_properties.h

## E.25 BiometricEvaluation::Image::Raw Class Reference

An image with no encoding or compression.

```
#include <be_image_raw.h>
```

Inheritance diagram for 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::Parameter-Error)  
*Accessor for decompressed data in grayscale.*

### E.25.1 Detailed Description

An image with no encoding or compression.

### E.25.2 Member Function Documentation

#### E.25.2.1 [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.25.2.2 [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

<a href="#"><i>Error::DataError</i></a>	<a href="#">Error</a> decompressing image data.
---	---

Implements [BiometricEvaluation::Image::Image](#).

**E.25.2.3** `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

<i>depth</i>	The desired bit depth of the resulting raw image. This value may either be 8 or 1.
--------------	--

## Returns

[Raw](#) image buffer.

## Exceptions

<a href="#"><i>Error::DataError</i></a>	<a href="#">Error</a> decompressing image data.
<a href="#"><i>Error::ParameterError</i></a>	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).

Implements [BiometricEvaluation::Image::Image](#).

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

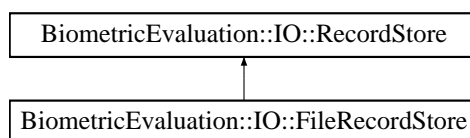
- `be_image_raw.h`

## E.26 BiometricEvaluation::IO::RecordStore Class Reference

A class to represent a data storage mechanism.

```
#include <be_io_recordstore.h>
```

Inheritance diagram for `BiometricEvaluation::IO::RecordStore`:



## 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 **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 void **replace** (const string &key, const void \*const data, const uint64\_t size)=0 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)
- 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 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

## E.26.1 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.26.2 Constructor & Destructor Documentation

**E.26.2.1** [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	<i>name</i>	The name of the <a href="#">RecordStore</a> to be created.
in	<i>description</i>	The text used to describe the store.
in	<i>type</i>	The type of <a href="#">RecordStore</a> .
in	<i>parentDir</i>	Where, in the file system, the store is to be rooted. This directory must exist.

### Returns

An object representing the new, empty store.

### Exceptions

<a href="#">Error::ObjectExists</a>	The store was previously created, or the directory where it would be created exists.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying storage system, or the the name malformed.

**E.26.2.2** `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	<i>name</i>	The name of the store to be opened.
in	<i>parentDir</i>	Where, in the file system, the store is rooted.
in	<i>mode</i>	The type of access a client of this <a href="#">RecordStore</a> has.

#### Returns

An object representing the existing store.

#### Exceptions

<a href="#">Error::ObjectDoesNotExist</a>	The <a href="#">RecordStore</a> does not exist.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying storage system, or the name is malformed.

### E.26.3 Member Function Documentation

**E.26.3.1** `string BiometricEvaluation::IO::RecordStore::getName ( ) const`

Return the name of the [RecordStore](#).

#### Returns

The [RecordStore](#)'s name.

**E.26.3.2** `string BiometricEvaluation::IO::RecordStore::getDescription ( ) const`

Obtain a textual description of the [RecordStore](#).

#### Returns

The [RecordStore](#)'s description.

**E.26.3.3** `unsigned int BiometricEvaluation::IO::RecordStore::getCount ( ) const`

Obtain the number of items in the [RecordStore](#).

#### Returns

The number of items in the [RecordStore](#).

**E.26.3.4** `virtual void BiometricEvaluation::IO::RecordStore::changeName ( const string & name ) throw (Error::ObjectExists, Error::StrategyError) [virtual]`

Change the name of the [RecordStore](#).

#### Parameters

in	<i>name</i>	The new name for the <a href="#">RecordStore</a> .
----	-------------	--



**Exceptions**

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system, or the name is malformed.
---	---

Reimplemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.5** `virtual void BiometricEvaluation::IO::RecordStore::changeDescription ( const string & description ) throw (Error::StrategyError) [virtual]`

Change the description of the [RecordStore](#).

**Parameters**

<code>in</code>	<i>description</i>	The new description.
-----------------	--------------------	----------------------

**Exceptions**

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system.
---	---

**E.26.3.6** `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**

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system.
---	---

Reimplemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.7** `virtual void BiometricEvaluation::IO::RecordStore::sync ( ) const throw (Error::StrategyError) [virtual]`

Synchronize the entire record store to persistent storage.

**Exceptions**

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system.
---	---

**E.26.3.8** `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	<i>key</i>	The key of the record to be inserted.
in	<i>data</i>	The data for the record.
in	<i>size</i>	The size, in bytes, of the record.

### Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	A record with the given key is already present.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system.

Implemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.9** `virtual void BiometricEvaluation::IO::RecordStore::remove ( const string & key ) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]`

Remove a record from the store.

### Parameters

in	<i>key</i>	The key of the record to be removed.
----	------------	--------------------------------------

### Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	A record for the key does not exist.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system.

Implemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.10** `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	<i>key</i>	The key of the record to be read.
in	<i>data</i>	Pointer to where the data is to be written.

### Returns

The size of the record.

### Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	A record for the key does not exist.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying storage system.

Implemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.11** `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**

in	<i>key</i>	The key of the record to be replaced.
in	<i>data</i>	The data for the record.
in	<i>size</i>	The size of data.

**Exceptions**

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

Implemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.12** `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**

in	<i>key</i>	The key of the record.
----	------------	------------------------

**Returns**

The record length.

**Exceptions**

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

Implemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.13** `virtual void BiometricEvaluation::IO::RecordStore::flush ( const string & key ) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]`

Commit the record's data to storage.

**Parameters**

in	<i>key</i>	The key of the record to be flushed.
----	------------	--------------------------------------

**Exceptions**

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

Implemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.14** `virtual uint64_t BiometricEvaluation::IO::RecordStore::sequence ( string & key, void *const data = NULL, int cursor = BE_RECSTORE_SEQ_NEXT ) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure 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 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

<a href="#">Error::ObjectDoesNotExist</a>	A record for the key does not exist.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying storage system.

Implemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.15** `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 <a href="#">sequence()</a> .
----	-----	---

#### Exceptions

<a href="#">Error::ObjectDoesNotExist</a>	A record for the key does not exist.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying storage system.

Implemented in [BiometricEvaluation::IO::FileRecordStore](#).

**E.26.3.16** `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	<i>name</i>	The name of the store to be opened.
in	<i>parentDir</i>	Where, in the file system, the store is rooted.
in	<i>mode</i>	The type of access a client of this <a href="#">RecordStore</a> has.

**Returns**

An object representing the existing store.

**Exceptions**

<a href="#">Error::ObjectDoesNotExist</a>	The <a href="#">RecordStore</a> does not exist.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying storage system, or the name is malformed.

**E.26.3.17** `static tr1::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::createRecordStore ( const string & name, const string & description, const string & type, const string & destDir ) throw (Error::ObjectExists, Error::StrategyError) [static]`

Create a new [RecordStore](#) and return a managed pointer to the 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**

in	<i>name</i>	The name of the store to be created.
in	<i>description</i>	The description of the store to be created.
in	<i>type</i>	The type of the store to be created.
in	<i>destDir</i>	Where, in the file system, the store will be created.

**Returns**

An auto\_ptr to the object representing the created store.

**Exceptions**

<a href="#">Error::ObjectDoesNotExist</a>	The <a href="#">RecordStore</a> does not exist.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying storage system, or the name is malformed.

**E.26.3.18** `static void BiometricEvaluation::IO::RecordStore::removeRecordStore ( const string & name, const string & parentDir ) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]`

Remove a [RecordStore](#) by deleting all persistant data associated with the store.

**Parameters**

in	<i>name</i>	The name of the existing <a href="#">RecordStore</a> .
in	<i>parentDir</i>	Where, in the file system, the store is rooted.

**Exceptions**

<a href="#">Error::ObjectDoesNotExist</a>	A record with the given key does not exist.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying storage system.

**E.26.3.19** `static void BiometricEvaluation::IO::RecordStore::mergeRecordStores ( const string & mergedName, const string & mergedDescription, const string & parentDir, const string & type, RecordStore * recordStores[], size_t numRecordStores ) throw (Error::ObjectExists, Error::StrategyError) [static]`

Create a new [RecordStore](#) that contains the contents of several RecordStores.

#### Parameters

in	<i>mergedName</i>	The name of the new <a href="#">RecordStore</a> that will be created.
in	<i>mergedDescription</i>	The text used to describe the <a href="#">RecordStore</a> .
in	<i>parentDir</i>	Where, in the file system, the new store should be rooted.
in	<i>type</i>	The type of <a href="#">RecordStore</a> that mergedName should be.
in	<i>recordStores</i>	An array of RecordStore* that should be merged into mergedName.
in	<i>numRecordStores</i>	The number of RecordStore* in recordStores.

#### Exceptions

<a href="#">Error::ObjectExists</a>	A <a href="#">RecordStore</a> with mergedNamed in parentDir already exists.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying storage system.

**E.26.3.20** `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

in	<i>mergedName</i>	The name of the new <a href="#">RecordStore</a> that will be created.
in	<i>mergedDescription</i>	The text used to describe the <a href="#">RecordStore</a> .
in	<i>parentDir</i>	Where, in the file system, the new store should be rooted.
in	<i>type</i>	The type of <a href="#">RecordStore</a> that mergedName should be.
in	<i>recordStores</i>	An array of <a href="#">RecordStore</a> shared pointers, such as those returned from IO::Factory, that should be merged into mergedName.
in	<i>numRecordStores</i>	The number of RecordStore* in recordStores.

#### Exceptions

<a href="#">Error::ObjectExists</a>	A <a href="#">RecordStore</a> with mergedNamed in parentDir already exists.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying storage system.

## E.26.4 Member Data Documentation

**E.26.4.1** `const string BiometricEvaluation::IO::RecordStore::INVALIDKEYCHARS [static]`

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

**E.26.4.2** `const string BiometricEvaluation::IO::RecordStore::CONTROLFILENAME [static]`

The name of the control file, a properties list

**E.26.4.3** `const string BiometricEvaluation::IO::RecordStore::NAMEPROPERTY` `[static]`

Property key for name of the [RecordStore](#)

**E.26.4.4** `const string BiometricEvaluation::IO::RecordStore::DESCRIPTIONPROPERTY` `[static]`

Property key for description of the [RecordStore](#)

**E.26.4.5** `const string BiometricEvaluation::IO::RecordStore::COUNTPROPERTY` `[static]`

Property key for the number of store items

**E.26.4.6** `const string BiometricEvaluation::IO::RecordStore::TYPEPROPERTY` `[static]`

Property key for the type of [RecordStore](#)

**E.26.4.7** `const string BiometricEvaluation::IO::RecordStore::BERKELEYDBTYPE` `[static]`

DBRecordStore type

**E.26.4.8** `const string BiometricEvaluation::IO::RecordStore::ARCHIVETYPE` `[static]`

ArchiveRecordStore type

**E.26.4.9** `const string BiometricEvaluation::IO::RecordStore::FILETYPE` `[static]`

[FileRecordStore](#) type

**E.26.4.10** `const string BiometricEvaluation::IO::RecordStore::SQLITETYPE` `[static]`

SQLiteRecordStore type

**E.26.4.11** `const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_START = 1` `[static]`

Tell [sequence\(\)](#) to sequence from beginning

**E.26.4.12** `const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_NEXT = 2` `[static]`

Tell sequence to sequence from current position

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

- `be_io_recordstore.h`

## E.27 BiometricEvaluation::Image::Resolution Struct Reference

A structure to represent the resolution of an image.

```
#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.27.1 Detailed Description

A structure to represent the resolution of an image.

### E.27.2 Member Enumeration Documentation

#### E.27.2.1 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.27.3 Constructor & Destructor Documentation

#### E.27.3.1 BiometricEvaluation::Image::Resolution::Resolution ( const double [xRes](#) = 0.0, const double [yRes](#) = 0.0, const [Kind](#) [units](#) = [PPI](#) )

Create a [Resolution](#) struct.

#### Parameters

<a href="#">in</a>	<a href="#">xRes</a>	<a href="#">Resolution</a> along the X-axis
<a href="#">in</a>	<a href="#">yRes</a>	<a href="#">Resolution</a> along the Y-axis
<a href="#">in</a>	<a href="#">units</a>	Units in which <a href="#">xRes</a> and <a href="#">yRes</a> are represented



## E.27.4 Member Data Documentation

### E.27.4.1 double BiometricEvaluation::Image::Resolution::xRes

Resolution along the X-axis

### E.27.4.2 double BiometricEvaluation::Image::Resolution::yRes

Resolution along the Y-axis

### E.27.4.3 Kind BiometricEvaluation::Image::Resolution::units

Units in which xRes and yRes are represented

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

- be\_image.h

## E.28 FPVTE2012::ReturnStatus Struct Reference

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

```
#include <fpvte2012.h>
```

### Public Member Functions

- [ReturnStatus](#) (const [StatusCode::Kind](#) code, const string info="")  
*Create a [ReturnStatus](#) object.*

### Public Attributes

- [StatusCode::Kind](#) code
- string info

### E.28.1 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.28.2 Constructor & Destructor Documentation

#### E.28.2.1 FPVTE2012::ReturnStatus::ReturnStatus ( const StatusCode::Kind code, const string info = " " )

Create a [ReturnStatus](#) object.

**Parameters**

in	code	The return status code; required.
in	info	The optional information string.

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

- fpvte2012.h

**E.29 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 throw (Error::Exception)  
*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 uint32\_t 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 uint32\_t 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.29.1 Member Function Documentation

**E.29.1.1** `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

out	<i>nist_assigned_identifier</i>	An ID which identifies the SDK under test. Cannot be the empty string.
out	<i>email_address</i>	Point of contact email address. Cannot be the empty string.

**E.29.1.2** `virtual ReturnStatus FPVTE2012::SDKInterface::initEnrollmentTemplateExtraction ( const string & configuration_directory, const uint32_t 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  $n \geq 1$  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

in	<i>configuration_directory</i>	A read-only directory containing vendor-supplied configuration parameters or run-time data files.
in	<i>num_subjects</i>	The total number of subjects that will be used in the enrollment set.

### Returns

The object containing a required status code and optional information string.

### Exceptions

<i>Error::Exception</i>	There was an error processing this request, and the exception string may contain additional information.
-------------------------	--

**E.29.1.3** `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.

### Parameters

in	<i>fingers</i>	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	<i>enrollment_template</i>	The template to be added to the enrollment set, extracted from the finger images. The format is not regulated.
out	<i>output_features</i>	Additional information for the images, produced by the SDK. This information will be used in final results analysis.

**Returns**

The object containing a required status code and optional information string.

**Exceptions**

<i>Error::Exception</i>	There was an error processing this request, and the exception string may contain additional information.
-------------------------	--

**E.29.1.4** `virtual ReturnStatus FVPTE2012::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**

in	<i>configuration_directory</i>	A read-only directory containing vendor-supplied configuration
in	<i>enrollment_directory</i>	The top-level directory in which all enrollment data will reside. After this finalization step the directory and its contents will 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.
in	<i>blade_count</i>	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.
in	<i>blade_memory</i>	Amount of memory available on each blade in gigabytes.
in	<i>input_templates</i>	The input RecordStore containing the enrolled templates, opened IO::READONLY.

**Returns**

The object containing a required status code and optional information string.

**Exceptions**

<i>Error::Exception</i>	There was an error processing this request, and the exception string may contain additional information.
-------------------------	--

**E.29.1.5** `virtual ReturnStatus FVPTE2012::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	<i>configuration_directory</i>	A read-only directory containing vendor-supplied configuration
----	--------------------------------	--

**Returns**

The object containing a required status code and optional information string.

**Exceptions**

<i>Error::Exception</i>	There was an error processing this request, and the exception string may contain additional information.
-------------------------	--

**E.29.1.6** `virtual ReturnStatus FPVTE2012::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.

**Parameters**

in	<i>fingers</i>	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	<i>fingers</i>	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	<i>search_template</i>	The search template, extracted from the finger images. The format is not regulated. The memory is allocated in the AutoArray object.
out	<i>output_features</i>	Additional information for the images, produced by the SDK. This information will be used in final results analysis.

**Returns**

The object containing a required status code and optional information string.

**Exceptions**

<i>Error::Exception</i>	There was an error processing this request, and the exception string may contain additional information.
-------------------------	--

**E.29.1.7** `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	<i>configuration_directory</i>	A read-only directory containing vendor-supplied configuration
in	<i>enrollment_directory</i>	The top-level directory in which all finalized enrollment data resides. The directory will have read-only access.
in	<i>blade_number</i>	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**

<i>Error::Exception</i>	There was an error processing this request, and the exception string may contain additional information.
-------------------------	--

**E.29.1.8** `virtual ReturnStatus FPVTE2012::SDKInterface::identifyTemplateStageOne ( const uint32_t 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	<i>search_ID</i>	The ID of the search subject. This ID does not identify the subject it is merely a sequence number used to distinguish different searches performed by the system and it will be used as the input to <i>identifyTemplateStageTwo</i> .
in	<i>search_template</i>	A template from <i>makeSearchTemplate</i> . The size is given by the <i>AutoArray</i> object.
in	<i>stage1_data_directory</i>	This directory will have read-write access. The output information from <i>identifyTemplateStageOne</i> that is needed in <i>indentifyTemplateStageTwo</i> 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**

<i>Error::Exception</i>	There was an error processing this request, and the exception string may contain additional information.
-------------------------	--

**E.29.1.9** `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	<i>configuration_directory</i>	A read-only directory containing vendor-supplied configuration
in	<i>enrollment_directory</i>	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**

<i>Error::Exception</i>	There was an error processing this request, and the exception string may contain additional information.
-------------------------	--

**E.29.1.10** `virtual ReturnStatus FPVTE2012::SDKInterface::identifyTemplateStageTwo ( const uint32_t 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	<i>search_ID</i>	The ID of the search subject. This ID does not identify the subject it is merely a sequence number used to distinguish different searches performed by the system.
in	<i>stage1_data_directory</i>	This directory will have read-only access and contains all the stored output data from the <code>identifyTemplateStageOne</code> processes for the search subject.
out	<i>candidates</i>	The candidate list returned from <code>identifyTemplateStageTwo</code> .

#### Returns

The object containing a required status code and optional information string.

#### Exceptions

<i>Error::Exception</i>	There was an error processing this request, and the exception string may contain additional information.
-------------------------	--

**E.29.1.11** `static SDKIpPtr FPVTE2012::SDKInterface::getSDK ( ) [static]`

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.30 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.30.1 Detailed Description

A structure to represent the size of an image, in pixels.

## E.30.2 Constructor & Destructor Documentation

### E.30.2.1 BiometricEvaluation::Image::Size::Size ( const uint32\_t xSize = 0, const uint32\_t ySize = 0 )

Create a [Size](#) struct.

#### Parameters

in	xSize	Number of pixels on the X-axis
in	ySize	Number of pixels on the Y-axis

## E.30.3 Member Data Documentation

### E.30.3.1 uint32\_t BiometricEvaluation::Image::Size::xSize

Number of pixels on the X-axis

### E.30.3.2 uint32\_t 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.31 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.31.1 Member Enumeration Documentation

#### E.31.1.1 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](#)

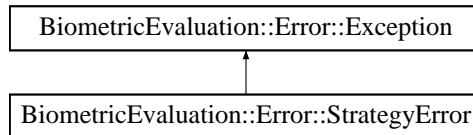


## E.32 BiometricEvaluation::Error::StrategyError Class Reference

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:



### Public Member Functions

- [StrategyError](#) ()
- [StrategyError](#) (string info)

#### E.32.1 Detailed Description

A [StrategyError](#) object is thrown when the underlying implementation of this interface encounters an error.

#### E.32.2 Constructor & Destructor Documentation

##### E.32.2.1 BiometricEvaluation::Error::StrategyError::StrategyError ( )

Construct a [StrategyError](#) object with the default information string.

##### Returns

The [StrategyError](#) object.

##### E.32.2.2 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