

BIOMETRIC EVALUATION COMMON FRAMEWORK

PROGRAMMER'S GUIDE

VERSION 0.1

WAYNE SALAMON
GREGORY FIUMARA

IMAGE GROUP
INFORMATION ACCESS DIVISION
INFORMATION TECHNOLOGY LABORATORY

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

JULY 30, 2012

Contents

1	Introduction	1
1.1	Rationale	1
2	Overview	3
3	Framework	5
4	Memory	7
4.1	AutoBuffer	7
4.2	AutoArray	8
4.3	IndexedBuffer	9
5	Error Handling	11
5.1	Biometric Evaluation Exceptions	11
5.2	Signal Handling	11
6	Input/Output	15
6.1	Utility	15
6.2	Record Management	15
6.3	Logging	17
6.4	Properties	18
6.5	Compressor	18
7	Time and Timing	21
7.1	Elapsed Time	21
7.2	Limiting Execution Time	21
8	Process Information	23
8.1	Process Statistics	23
8.2	Process Management	25
8.2.1	Manager	25
8.2.2	Worker	25
8.2.3	WorkerController	27
9	System	29
10	Image	31
10.1	The Image Namespace	31
10.2	The Image Class	31
10.3	Raw Image	32
10.4	JPEG	32
10.5	JPEGL	32

10.6	JPEG2000	32
10.7	NetPBM	33
10.8	PNG	33
10.9	WSQ	33
11	Text	35
12	Feature	37
12.1	ANSI/NIST Features	37
13	Finger	39
13.1	ANSI/NIST Minutiae Data Record	39
14	View	41
14.1	Finger Views	41
14.1.1	ANSI/NIST Finger Views	41
14.1.2	ISO/INCITS Finger Views	42
15	Data Interchange	43
15.1	ANSI/NIST Data Records	43
15.2	INCITS Data Records	46
15.2.1	Finger Views	46
A	Namespace Index	51
A.1	Namespace List	51
B	Class Index	53
B.1	Class Hierarchy	53
C	Class Index	57
C.1	Class List	57
D	Namespace Documentation	61
D.1	BiometricEvaluation::Error Namespace Reference	61
D.1.1	Detailed Description	62
D.1.2	Function Documentation	62
D.1.2.1	errorStr	62
D.2	BiometricEvaluation::Finger Namespace Reference	62
D.2.1	Detailed Description	63
D.2.2	Function Documentation	64
D.2.2.1	operator<<	64
D.3	BiometricEvaluation::Framework Namespace Reference	64
D.3.1	Detailed Description	64
D.3.2	Function Documentation	64
D.3.2.1	getMajorVersion	64
D.3.2.2	getMinorVersion	65
D.3.2.3	getCompiler	65
D.3.2.4	getCompileDate	65
D.3.2.5	getCompileTime	65
D.3.2.6	getCompilerVersion	65
D.4	BiometricEvaluation::Image Namespace Reference	65
D.4.1	Detailed Description	67
D.4.2	Function Documentation	67

D.4.2.1	operator<<	67
D.4.2.2	distance	67
D.5	BiometricEvaluation::IO Namespace Reference	67
D.5.1	Detailed Description	68
D.5.2	Typedef Documentation	68
D.5.2.1	ManifestMap	68
D.5.2.2	PropertiesMap	68
D.6	BiometricEvaluation::IO::Utility Namespace Reference	69
D.6.1	Detailed Description	69
D.6.2	Function Documentation	70
D.6.2.1	removeDirectory	70
D.6.2.2	removeDirectory	70
D.6.2.3	copyDirectoryContents	70
D.6.2.4	setAsideName	71
D.6.2.5	getFileSize	71
D.6.2.6	fileExists	72
D.6.2.7	validateRootName	72
D.6.2.8	constructAndCheckPath	72
D.6.2.9	makePath	73
D.6.2.10	readFile	73
D.6.2.11	writeFile	73
D.6.2.12	writeFile	74
D.6.2.13	isReadable	74
D.6.2.14	isWritable	74
D.6.2.15	createTemporaryFile	75
D.6.2.16	createTemporaryFile	75
D.7	BiometricEvaluation::Memory Namespace Reference	76
D.7.1	Detailed Description	76
D.8	BiometricEvaluation::Process Namespace Reference	76
D.8.1	Detailed Description	77
D.8.2	Typedef Documentation	77
D.8.2.1	ParameterList	77
D.9	BiometricEvaluation::System Namespace Reference	77
D.9.1	Detailed Description	78
D.9.2	Function Documentation	78
D.9.2.1	getCPUCount	78
D.9.2.2	getRealMemorySize	78
D.9.2.3	getLoadAverage	78
D.10	BiometricEvaluation::Text Namespace Reference	79
D.10.1	Detailed Description	79
D.10.2	Function Documentation	79
D.10.2.1	digest	79
D.10.2.2	digest	80
D.10.2.3	split	80
D.10.2.4	filename	80
D.10.2.5	dirname	81
D.11	BiometricEvaluation::Time Namespace Reference	81
D.11.1	Detailed Description	81
D.12	BiometricEvaluation::View Namespace Reference	82
D.12.1	Detailed Description	82
D.12.2	Function Documentation	82

D.12.2.1	operator<<	82
E	Class Documentation	83
E.1	BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged Class Reference	83
E.1.1	Detailed Description	83
E.1.2	Member Enumeration Documentation	83
E.1.2.1	Kind	83
E.2	BiometricEvaluation::Feature::AN2K7Minutiae Class Reference	84
E.2.1	Detailed Description	85
E.2.2	Constructor & Destructor Documentation	85
E.2.2.1	AN2K7Minutiae	85
E.2.2.2	AN2K7Minutiae	86
E.2.3	Member Function Documentation	86
E.2.3.1	convertPatternClassification	86
E.2.3.2	convertPatternClassification	86
E.2.3.3	convertEncodingMethod	87
E.2.3.4	getPatternClassificationSet	87
E.2.3.5	getOriginatingFingerprintReadingSystem	87
E.2.3.6	convertCoordinate	87
E.3	BiometricEvaluation::Finger::AN2KMinutiaeDataRecord Class Reference	88
E.3.1	Detailed Description	88
E.3.2	Constructor & Destructor Documentation	88
E.3.2.1	AN2KMinutiaeDataRecord	88
E.3.2.2	AN2KMinutiaeDataRecord	89
E.3.3	Member Function Documentation	89
E.3.3.1	getAN2K7Minutiae	89
E.3.3.2	getImpressionType	89
E.3.3.3	getRegisteredVendorBlock	90
E.4	BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric Struct Reference	90
E.4.1	Detailed Description	90
E.5	BiometricEvaluation::DataInterchange::AN2KRecord Class Reference	91
E.5.1	Detailed Description	92
E.5.2	Member Typedef Documentation	92
E.5.2.1	DomainName	92
E.5.2.2	CharacterSet	92
E.5.3	Constructor & Destructor Documentation	92
E.5.3.1	AN2KRecord	92
E.5.3.2	AN2KRecord	93
E.5.4	Member Function Documentation	93
E.5.4.1	recordLocations	93
E.5.4.2	recordLocations	93
E.5.4.3	getVersionNumber	94
E.5.4.4	getDate	94
E.5.4.5	getDestinationAgency	94
E.5.4.6	getOriginatingAgency	94
E.5.4.7	getTransactionControlNumber	94
E.5.4.8	getNativeScanningResolution	95
E.5.4.9	getNominalTransmittingResolution	95
E.5.4.10	getFingerLatentCount	95
E.5.4.11	getFingerLatents	95
E.5.4.12	getFingerCaptureCount	95
E.5.4.13	getFingerCaptures	96

	E.5.4.14	getMinutiaeDataRecordSet	96
	E.5.4.15	getPriority	96
	E.5.4.16	getDomainName	96
	E.5.4.17	getGreenwichMeanTime	96
	E.5.4.18	getDirectoryOfCharacterSets	97
E.6		BiometricEvaluation::Finger::AN2KView Class Reference	97
E.6.1		Detailed Description	98
E.6.2		Constructor & Destructor Documentation	98
	E.6.2.1	AN2KView	98
E.6.3		Member Function Documentation	98
	E.6.3.1	convertPosition	98
	E.6.3.2	populateFGP	99
	E.6.3.3	convertFingerImageCode	99
	E.6.3.4	getMinutiaeDataRecordSet	99
	E.6.3.5	getPositions	100
E.7		BiometricEvaluation::View::AN2KView Class Reference	100
E.7.1		Detailed Description	102
E.7.2		Constructor & Destructor Documentation	102
	E.7.2.1	AN2KView	102
	E.7.2.2	AN2KView	102
E.7.3		Member Function Documentation	102
	E.7.3.1	convertDeviceMonitoringMode	102
	E.7.3.2	convertCompressionAlgorithm	103
	E.7.3.3	getImage	103
	E.7.3.4	getImageSize	103
	E.7.3.5	getImageResolution	103
	E.7.3.6	getImageDepth	104
	E.7.3.7	getCompressionAlgorithm	104
	E.7.3.8	getScanResolution	104
	E.7.3.9	getMinutiaeDataRecordSet	104
	E.7.3.10	getRecordType	104
	E.7.3.11	getAN2KRecord	105
E.8		BiometricEvaluation::Finger::AN2KViewCapture Class Reference	105
E.8.1		Detailed Description	106
E.8.2		Constructor & Destructor Documentation	106
	E.8.2.1	AN2KViewCapture	106
	E.8.2.2	AN2KViewCapture	107
E.8.3		Member Function Documentation	107
	E.8.3.1	convertAmputatedBandaged	107
	E.8.3.2	convertFingerSegmentPosition	107
	E.8.3.3	convertAlternateFingerSegmentPosition	108
	E.8.3.4	extractNISTQuality	108
	E.8.3.5	getNISTQualityMetric	108
	E.8.3.6	getSegmentationQualityMetric	109
	E.8.3.7	getAmputatedBandaged	109
	E.8.3.8	getFingerSegmentPositionSet	109
	E.8.3.9	getAlternateFingerSegmentPositionSet	109
	E.8.3.10	getFingerprintQualityMetric	109
E.9		BiometricEvaluation::Finger::AN2KViewFixedResolution Class Reference	109
E.9.1		Detailed Description	110
E.9.2		Constructor & Destructor Documentation	110

E.9.2.1	AN2KViewFixedResolution	110
E.10	BiometricEvaluation::Finger::AN2KViewLatent Class Reference	111
E.10.1	Constructor & Destructor Documentation	111
E.10.1.1	AN2KViewLatent	111
E.10.1.2	AN2KViewLatent	111
E.10.2	Member Function Documentation	112
E.10.2.1	getLatentQualityMetric	112
E.11	BiometricEvaluation::Finger::AN2KViewVariableResolution Class Reference	112
E.11.1	Detailed Description	113
E.11.2	Constructor & Destructor Documentation	113
E.11.2.1	AN2KViewVariableResolution	113
E.11.2.2	AN2KViewVariableResolution	113
E.11.3	Member Function Documentation	114
E.11.3.1	getPositions	114
E.11.3.2	getPrintPositionCoordinates	114
E.11.3.3	convertPrintPositionCoordinate	114
E.11.3.4	parsePositionDescriptors	114
E.12	BiometricEvaluation::View::AN2KViewVariableResolution Class Reference	115
E.12.1	Detailed Description	116
E.12.2	Constructor & Destructor Documentation	116
E.12.2.1	AN2KViewVariableResolution	116
E.12.2.2	AN2KViewVariableResolution	116
E.12.3	Member Function Documentation	117
E.12.3.1	extractQuality	117
E.12.3.2	getComment	117
E.12.3.3	getUserDefinedField	117
E.12.3.4	parseUserDefinedField	118
E.12.3.5	getQualityMetric	118
E.13	BiometricEvaluation::Finger::ANSI2004View Class Reference	118
E.13.1	Detailed Description	119
E.13.2	Constructor & Destructor Documentation	119
E.13.2.1	ANSI2004View	119
E.13.2.2	ANSI2004View	120
E.13.3	Member Function Documentation	120
E.13.3.1	readCoreDeltaData	120
E.14	BiometricEvaluation::Finger::ANSI2007View Class Reference	120
E.14.1	Detailed Description	122
E.14.2	Constructor & Destructor Documentation	122
E.14.2.1	ANSI2007View	122
E.14.2.2	ANSI2007View	122
E.14.3	Member Function Documentation	122
E.14.3.1	readFMRHeader	123
E.14.3.2	readFVMR	123
E.14.3.3	readCoreDeltaData	123
E.15	BiometricEvaluation::IO::ArchiveRecordStore Class Reference	124
E.15.1	Detailed Description	125
E.15.2	Constructor & Destructor Documentation	125
E.15.2.1	ArchiveRecordStore	125
E.15.2.2	ArchiveRecordStore	126
E.15.2.3	~ArchiveRecordStore	126
E.15.3	Member Function Documentation	126

E.15.3.1	getSpaceUsed	126
E.15.3.2	sync	126
E.15.3.3	insert	127
E.15.3.4	remove	127
E.15.3.5	read	127
E.15.3.6	replace	128
E.15.3.7	length	128
E.15.3.8	flush	129
E.15.3.9	sequence	129
E.15.3.10	setCursorAtKey	130
E.15.3.11	changeName	130
E.15.3.12	needsVacuum	130
E.15.3.13	needsVacuum	131
E.15.3.14	vacuum	131
E.15.3.15	getArchiveName	132
E.15.3.16	getManifestName	132
E.16	BiometricEvaluation::Memory::AutoArray< T > Class Template Reference	132
E.16.1	Detailed Description	133
E.16.2	Member Typedef Documentation	133
E.16.2.1	value_type	133
E.16.2.2	size_type	133
E.16.2.3	iterator	134
E.16.2.4	const_iterator	134
E.16.2.5	reference	134
E.16.2.6	const_reference	134
E.16.3	Constructor & Destructor Documentation	134
E.16.3.1	AutoArray	134
E.16.3.2	AutoArray	134
E.16.3.3	~AutoArray	135
E.16.4	Member Function Documentation	135
E.16.4.1	operator T *	135
E.16.4.2	operator const T *	135
E.16.4.3	operator[]	135
E.16.4.4	operator[]	135
E.16.4.5	at	136
E.16.4.6	at	136
E.16.4.7	begin	136
E.16.4.8	begin	137
E.16.4.9	end	137
E.16.4.10	end	137
E.16.4.11	size	137
E.16.4.12	resize	137
E.16.4.13	copy	138
E.16.4.14	copy	138
E.16.4.15	operator=	138
E.17	BiometricEvaluation::Memory::AutoBuffer< T > Class Template Reference	139
E.17.1	Member Typedef Documentation	139
E.17.1.1	value_type	139
E.18	BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet Struct Reference	140
E.18.1	Constructor & Destructor Documentation	140
E.18.1.1	CharacterSet	140

E.18.2	Member Data Documentation	140
E.18.2.1	identifier	140
E.18.2.2	commonName	140
E.18.2.3	version	141
E.19	BiometricEvaluation::IO::CompressedRecordStore Class Reference	141
E.19.1	Detailed Description	142
E.19.2	Constructor & Destructor Documentation	142
E.19.2.1	CompressedRecordStore	142
E.19.2.2	CompressedRecordStore	142
E.19.2.3	CompressedRecordStore	143
E.19.3	Member Function Documentation	143
E.19.3.1	getSpaceUsed	143
E.19.3.2	sync	143
E.19.3.3	insert	144
E.19.3.4	remove	144
E.19.3.5	read	144
E.19.3.6	replace	145
E.19.3.7	length	145
E.19.3.8	flush	146
E.19.3.9	sequence	146
E.19.3.10	setCursorAtKey	147
E.19.3.11	changeName	147
E.19.4	Member Data Documentation	147
E.19.4.1	BACKING_STORE	147
E.19.4.2	COMPRESSOR_TYPE_KEY	147
E.20	BiometricEvaluation::Image::CompressionAlgorithm Class Reference	148
E.20.1	Detailed Description	148
E.21	BiometricEvaluation::IO::Compressor Class Reference	148
E.21.1	Detailed Description	150
E.21.2	Member Enumeration Documentation	150
E.21.2.1	Kind	150
E.21.3	Constructor & Destructor Documentation	150
E.21.3.1	Compressor	150
E.21.3.2	~Compressor	150
E.21.4	Member Function Documentation	150
E.21.4.1	kindToString	150
E.21.4.2	stringToKind	151
E.21.4.3	compress	151
E.21.4.4	compress	152
E.21.4.5	compress	152
E.21.4.6	compress	153
E.21.4.7	compress	153
E.21.4.8	compress	153
E.21.4.9	decompress	154
E.21.4.10	decompress	154
E.21.4.11	decompress	155
E.21.4.12	decompress	155
E.21.4.13	decompress	156
E.21.4.14	decompress	156
E.21.4.15	setOption	156
E.21.4.16	setOption	157

E.21.4.17	getOption	157
E.21.4.18	getOptionAsInteger	157
E.21.4.19	removeOption	158
E.21.4.20	createCompressor	158
E.21.5	Member Data Documentation	158
E.21.5.1	GZIPTYPE	158
E.22	BiometricEvaluation::Error::ConversionError Class Reference	159
E.22.1	Detailed Description	159
E.22.2	Constructor & Destructor Documentation	159
E.22.2.1	ConversionError	159
E.22.2.2	ConversionError	159
E.23	BiometricEvaluation::Image::Coordinate Struct Reference	160
E.23.1	Detailed Description	160
E.23.2	Constructor & Destructor Documentation	160
E.23.2.1	Coordinate	160
E.23.3	Member Data Documentation	160
E.23.3.1	x	160
E.23.3.2	y	160
E.23.3.3	xDistance	161
E.23.3.4	yDistance	161
E.24	BiometricEvaluation::Feature::CorePoint Struct Reference	161
E.24.1	Detailed Description	161
E.25	BiometricEvaluation::Error::DataError Class Reference	161
E.25.1	Detailed Description	162
E.25.2	Constructor & Destructor Documentation	162
E.25.2.1	DataError	162
E.25.2.2	DataError	162
E.26	BiometricEvaluation::IO::DBRecordStore Class Reference	162
E.26.1	Detailed Description	163
E.26.2	Constructor & Destructor Documentation	163
E.26.2.1	DBRecordStore	163
E.26.2.2	DBRecordStore	164
E.26.3	Member Function Documentation	164
E.26.3.1	getSpaceUsed	164
E.26.3.2	sync	164
E.26.3.3	insert	165
E.26.3.4	remove	165
E.26.3.5	read	165
E.26.3.6	replace	166
E.26.3.7	length	166
E.26.3.8	flush	167
E.26.3.9	sequence	167
E.26.3.10	setCursorAtKey	168
E.26.3.11	changeName	168
E.27	BiometricEvaluation::Feature::DeltaPoint Struct Reference	168
E.27.1	Detailed Description	169
E.28	BiometricEvaluation::View::AN2KView::DeviceMonitoringMode Class Reference	169
E.28.1	Detailed Description	169
E.28.2	Member Enumeration Documentation	169
E.28.2.1	Kind	169
E.29	BiometricEvaluation::DataInterchange::AN2KRecord::DomainName Struct Reference	170

E.29.1 Detailed Description	170
E.29.2 Constructor & Destructor Documentation	170
E.29.2.1 DomainName	170
E.29.3 Member Data Documentation	170
E.29.3.1 identifier	170
E.29.3.2 version	171
E.30 BiometricEvaluation::Feature::AN2K7Minutiae::EncodingMethod Class Reference	171
E.30.1 Detailed Description	171
E.31 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry Struct Reference	171
E.31.1 Constructor & Destructor Documentation	171
E.31.1.1 Entry	171
E.31.2 Member Data Documentation	172
E.31.2.1 standard	172
E.31.2.2 code	172
E.32 BiometricEvaluation::Error::Exception Class Reference	172
E.32.1 Detailed Description	173
E.32.2 Constructor & Destructor Documentation	173
E.32.2.1 Exception	173
E.32.2.2 Exception	174
E.32.3 Member Function Documentation	174
E.32.3.1 getInfo	174
E.33 BiometricEvaluation::Error::FileError Class Reference	174
E.33.1 Detailed Description	175
E.33.2 Constructor & Destructor Documentation	175
E.33.2.1 FileError	175
E.33.2.2 FileError	175
E.34 BiometricEvaluation::IO::FileRecordStore Class Reference	175
E.34.1 Detailed Description	176
E.34.2 Constructor & Destructor Documentation	176
E.34.2.1 FileRecordStore	176
E.34.2.2 FileRecordStore	177
E.34.3 Member Function Documentation	177
E.34.3.1 getSpaceUsed	177
E.34.3.2 insert	177
E.34.3.3 remove	178
E.34.3.4 read	178
E.34.3.5 replace	179
E.34.3.6 length	179
E.34.3.7 flush	179
E.34.3.8 sequence	180
E.34.3.9 setCursorAtKey	180
E.34.3.10 changeName	181
E.35 BiometricEvaluation::Finger::FingerImageCode Class Reference	181
E.35.1 Detailed Description	181
E.36 BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem Struct Reference	182
E.36.1 Detailed Description	182
E.36.2 Member Data Documentation	182
E.36.2.1 name	182
E.36.2.2 method	182
E.36.2.3 equipment	182
E.37 BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition Struct Reference	182

E.37.1	Detailed Description	183
E.37.2	Constructor & Destructor Documentation	183
E.37.2.1	FingerSegmentPosition	183
E.37.3	Member Data Documentation	183
E.37.3.1	fingerPosition	183
E.37.3.2	coordinates	183
E.38	BiometricEvaluation::Process::ForkManager Class Reference	183
E.38.1	Detailed Description	184
E.38.2	Constructor & Destructor Documentation	184
E.38.2.1	ForkManager	184
E.38.3	Member Function Documentation	185
E.38.3.1	getNumCompletedWorkers	185
E.38.3.2	getNumActiveWorkers	185
E.38.3.3	getTotalWorkers	185
E.38.3.4	addWorker	185
E.38.3.5	startWorkers	186
E.38.3.6	startWorker	186
E.38.3.7	stopWorker	186
E.38.3.8	reset	187
E.38.4	Member Data Documentation	187
E.38.4.1	_workers	187
E.39	BiometricEvaluation::Process::ForkWorkerController Class Reference	187
E.39.1	Detailed Description	188
E.39.2	Member Function Documentation	188
E.39.2.1	isWorking	188
E.39.2.2	reset	189
E.39.2.3	getPID	189
E.39.2.4	_stop	189
E.39.3	Friends And Related Function Documentation	189
E.39.3.1	ForkManager::startWorkers	189
E.39.3.2	ForkManager::startWorker	190
E.39.3.3	ForkManager::stopWorker	190
E.39.3.4	ForkManager::addWorker	191
E.40	BiometricEvaluation::IO::GZip Class Reference	191
E.40.1	Detailed Description	192
E.40.2	Member Function Documentation	192
E.40.2.1	compress	192
E.40.2.2	compress	193
E.40.2.3	compress	193
E.40.2.4	compress	194
E.40.2.5	compress	194
E.40.2.6	compress	195
E.40.2.7	decompress	195
E.40.2.8	decompress	195
E.40.2.9	decompress	196
E.40.2.10	decompress	196
E.40.2.11	decompress	197
E.40.2.12	decompress	197
E.40.3	Member Data Documentation	198
E.40.3.1	COMPRESSION_LEVEL	198
E.40.3.2	COMPRESSION_STRATEGY	198

E.40.3.3	COMPRESSION_METHOD	198
E.40.3.4	INPUT_DATA_TYPE	198
E.40.3.5	WINDOW_BITS	198
E.40.3.6	MEMORY_LEVEL	198
E.40.3.7	CHUNK_SIZE	198
E.41	BiometricEvaluation::Image::Image Class Reference	198
E.41.1	Detailed Description	200
E.41.2	Constructor & Destructor Documentation	200
E.41.2.1	Image	200
E.41.2.2	Image	201
E.41.3	Member Function Documentation	201
E.41.3.1	getCompressionAlgorithm	201
E.41.3.2	getResolution	201
E.41.3.3	getData	201
E.41.3.4	getRawData	201
E.41.3.5	getRawGrayscaleData	202
E.41.3.6	getDimensions	202
E.41.3.7	getDepth	203
E.41.3.8	valueInColorspace	203
E.41.3.9	openImage	203
E.41.3.10	openImage	203
E.41.3.11	openImage	204
E.41.3.12	getCompressionAlgorithm	204
E.41.3.13	getCompressionAlgorithm	205
E.41.3.14	getCompressionAlgorithm	205
E.41.3.15	setResolution	206
E.41.3.16	setDimensions	206
E.41.3.17	setDepth	206
E.41.4	Member Data Documentation	206
E.41.4.1	bitsPerComponent	206
E.41.4.2	_raw_data	206
E.42	BiometricEvaluation::Finger::Impression Class Reference	206
E.42.1	Detailed Description	207
E.43	BiometricEvaluation::Feature::INCITSMinutiae Class Reference	207
E.43.1	Detailed Description	209
E.43.2	Constructor & Destructor Documentation	209
E.43.2.1	INCITSMinutiae	209
E.43.3	Member Function Documentation	209
E.43.3.1	setMinutiaPoints	209
E.43.3.2	setRidgeCountItems	209
E.43.3.3	setCorePointSet	210
E.43.3.4	setDeltaPointSet	210
E.44	BiometricEvaluation::Finger::INCITSView Class Reference	210
E.44.1	Detailed Description	213
E.44.2	Constructor & Destructor Documentation	213
E.44.2.1	INCITSView	213
E.44.2.2	INCITSView	213
E.44.3	Member Function Documentation	214
E.44.3.1	convertPosition	214
E.44.3.2	convertImpression	214
E.44.3.3	getPosition	214

E.44.3.4	getImpressionType	215
E.44.3.5	getQuality	215
E.44.3.6	getCaptureEquipmentID	215
E.44.3.7	isAppendixFCompliant	215
E.44.3.8	getImage	215
E.44.3.9	getImageSize	215
E.44.3.10	getImageResolution	216
E.44.3.11	getImageDepth	216
E.44.3.12	getCompressionAlgorithm	216
E.44.3.13	getScanResolution	216
E.44.3.14	getFMRData	216
E.44.3.15	getFIRData	216
E.44.3.16	setMinutiaeData	217
E.44.3.17	setPosition	217
E.44.3.18	setImpressionType	217
E.44.3.19	setQuality	217
E.44.3.20	setViewNumber	217
E.44.3.21	setCaptureEquipmentID	218
E.44.3.22	setCBEFFProductIDs	218
E.44.3.23	setAppendixFCompliance	218
E.44.3.24	setImageSize	218
E.44.3.25	setImageResolution	218
E.44.3.26	setScanResolution	219
E.44.3.27	setImageData	219
E.44.3.28	readFMRHeader	219
E.44.3.29	readFVMR	220
E.44.3.30	readMinutiaeDataPoints	220
E.44.3.31	readExtendedDataBlock	220
E.44.3.32	readRidgeCountData	221
E.44.3.33	readCoreDeltaData	221
E.45	BiometricEvaluation::Memory::IndexedBuffer Class Reference	221
E.45.1	Detailed Description	222
E.45.2	Constructor & Destructor Documentation	223
E.45.2.1	IndexedBuffer	223
E.45.3	Member Function Documentation	223
E.45.3.1	getSize	223
E.45.3.2	getIndex	223
E.45.3.3	setIndex	223
E.45.3.4	scanU8Val	223
E.45.3.5	scanU16Val	224
E.45.3.6	scanBeU16Val	224
E.45.3.7	scanU32Val	224
E.45.3.8	scanBeU32Val	224
E.45.3.9	scanU64Val	225
E.45.3.10	scan	225
E.45.3.11	operator[]	225
E.45.3.12	operator[]	226
E.46	BiometricEvaluation::Finger::ISO2005View Class Reference	226
E.46.1	Detailed Description	227
E.46.2	Constructor & Destructor Documentation	227
E.46.2.1	ISO2005View	227

E.46.2.2	ISO2005View	227
E.46.3	Member Function Documentation	228
E.46.3.1	readCoreDeltaData	228
E.47	BiometricEvaluation::Image::JPEG Class Reference	228
E.47.1	Detailed Description	229
E.47.2	Member Function Documentation	229
E.47.2.1	getRawGrayscaleData	229
E.47.2.2	getRawData	230
E.47.2.3	isJPEG	230
E.48	BiometricEvaluation::Image::JPEG2000 Class Reference	230
E.48.1	Detailed Description	231
E.48.2	Constructor & Destructor Documentation	231
E.48.2.1	JPEG2000	231
E.48.3	Member Function Documentation	231
E.48.3.1	getRawData	231
E.48.3.2	getRawGrayscaleData	232
E.48.3.3	isJPEG2000	232
E.49	BiometricEvaluation::Image::JPEGL Class Reference	233
E.49.1	Detailed Description	233
E.49.2	Member Function Documentation	233
E.49.2.1	getRawGrayscaleData	233
E.49.2.2	getRawData	234
E.49.2.3	isJPEGL	234
E.50	BiometricEvaluation::IO::LogCabinet Class Reference	235
E.50.1	Detailed Description	235
E.50.2	Constructor & Destructor Documentation	235
E.50.2.1	LogCabinet	235
E.50.2.2	LogCabinet	236
E.50.3	Member Function Documentation	236
E.50.3.1	newLogSheet	236
E.50.3.2	getName	236
E.50.3.3	getDescription	237
E.50.3.4	getCount	237
E.50.3.5	remove	237
E.51	BiometricEvaluation::IO::LogSheet Class Reference	237
E.51.1	Detailed Description	239
E.51.2	Constructor & Destructor Documentation	239
E.51.2.1	LogSheet	239
E.51.2.2	LogSheet	240
E.51.2.3	~LogSheet	240
E.51.2.4	LogSheet	240
E.51.3	Member Function Documentation	240
E.51.3.1	write	240
E.51.3.2	writeComment	241
E.51.3.3	newEntry	241
E.51.3.4	getCurrentEntry	241
E.51.3.5	resetCurrentEntry	242
E.51.3.6	getCurrentEntryNumber	242
E.51.3.7	sync	242
E.51.3.8	setAutoSync	242
E.51.3.9	sequence	242

E.51.3.10 trim	243
E.51.3.11 mergeLogSheets	243
E.51.3.12 operator=	243
E.51.3.13 updateCursor	244
E.51.4 Member Data Documentation	244
E.51.4.1 CommentDelimiter	244
E.51.4.2 EntryDelimiter	244
E.51.4.3 DescriptionTag	244
E.51.4.4 BE_LOGSHEET_SEQ_START	244
E.51.4.5 BE_LOGSHEET_SEQ_NEXT	244
E.51.4.6 _entryNumber	244
E.51.4.7 _theLogFile	244
E.51.4.8 _autoSync	244
E.51.4.9 _sequenceFile	245
E.51.4.10 _cursor	245
E.52 BiometricEvaluation::Process::Manager Class Reference	245
E.52.1 Detailed Description	246
E.52.2 Member Function Documentation	246
E.52.2.1 addWorker	246
E.52.2.2 getNumCompletedWorkers	246
E.52.2.3 getNumActiveWorkers	247
E.52.2.4 getTotalWorkers	247
E.52.2.5 startWorkers	247
E.52.2.6 startWorker	248
E.52.2.7 reset	248
E.52.2.8 stopWorker	248
E.53 BiometricEvaluation::IO::ManifestEntry Struct Reference	249
E.53.1 Detailed Description	249
E.53.2 Member Data Documentation	249
E.53.2.1 offset	249
E.53.2.2 size	249
E.54 BiometricEvaluation::Error::MemoryError Class Reference	249
E.54.1 Detailed Description	250
E.54.2 Constructor & Destructor Documentation	250
E.54.2.1 MemoryError	250
E.54.2.2 MemoryError	250
E.55 BiometricEvaluation::Feature::Minutiae Class Reference	250
E.55.1 Detailed Description	251
E.56 BiometricEvaluation::Feature::MinutiaeFormat Class Reference	251
E.56.1 Detailed Description	251
E.57 BiometricEvaluation::Feature::MinutiaeType Class Reference	252
E.57.1 Detailed Description	252
E.58 BiometricEvaluation::Feature::MinutiaPoint Struct Reference	252
E.58.1 Detailed Description	252
E.59 BiometricEvaluation::Image::NetPBM Class Reference	252
E.59.1 Detailed Description	253
E.59.2 Member Function Documentation	254
E.59.2.1 getRawData	254
E.59.2.2 getRawGrayscaleData	254
E.59.2.3 isNetPBM	255
E.59.2.4 skipLine	255

E.59.2.5	skipComment	255
E.59.2.6	getNextValue	256
E.59.2.7	ASCIIBitmapTo8Bit	256
E.59.2.8	ASCIIPixmapToBinaryPixmap	256
E.59.2.9	BinaryBitmapTo8Bit	257
E.60	BiometricEvaluation::Error::NotImplemented Class Reference	257
E.60.1	Detailed Description	258
E.60.2	Constructor & Destructor Documentation	258
E.60.2.1	NotImplemented	258
E.60.2.2	NotImplemented	258
E.61	BiometricEvaluation::Error::ObjectDoesNotExist Class Reference	258
E.61.1	Detailed Description	259
E.61.2	Constructor & Destructor Documentation	259
E.61.2.1	ObjectDoesNotExist	259
E.61.2.2	ObjectDoesNotExist	259
E.62	BiometricEvaluation::Error::ObjectExists Class Reference	259
E.62.1	Detailed Description	260
E.62.2	Constructor & Destructor Documentation	260
E.62.2.1	ObjectExists	260
E.62.2.2	ObjectExists	260
E.63	BiometricEvaluation::Error::ObjectIsClosed Class Reference	260
E.63.1	Detailed Description	260
E.63.2	Constructor & Destructor Documentation	261
E.63.2.1	ObjectIsClosed	261
E.63.2.2	ObjectIsClosed	261
E.64	BiometricEvaluation::Error::ObjectIsOpen Class Reference	261
E.64.1	Detailed Description	261
E.64.2	Constructor & Destructor Documentation	262
E.64.2.1	ObjectIsOpen	262
E.64.2.2	ObjectIsOpen	262
E.65	BiometricEvaluation::Error::ParameterError Class Reference	262
E.65.1	Detailed Description	262
E.65.2	Constructor & Destructor Documentation	263
E.65.2.1	ParameterError	263
E.65.2.2	ParameterError	263
E.66	BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification Class Reference	263
E.66.1	Detailed Description	263
E.67	BiometricEvaluation::Finger::PatternClassification Class Reference	264
E.67.1	Detailed Description	264
E.68	BiometricEvaluation::Image::PNG Class Reference	264
E.68.1	Detailed Description	265
E.68.2	Member Function Documentation	265
E.68.2.1	getRawData	265
E.68.2.2	getRawGrayscaleData	265
E.68.2.3	isPNG	266
E.69	BiometricEvaluation::Finger::Position Class Reference	266
E.69.1	Detailed Description	266
E.70	BiometricEvaluation::Process::POSIXThreadManager Class Reference	267
E.70.1	Detailed Description	267
E.70.2	Constructor & Destructor Documentation	268
E.70.2.1	POSIXThreadManager	268

E.70.3	Member Function Documentation	268
E.70.3.1	getNumCompletedWorkers	268
E.70.3.2	getNumActiveWorkers	268
E.70.3.3	getTotalWorkers	268
E.70.3.4	addWorker	269
E.70.3.5	startWorkers	269
E.70.3.6	startWorker	269
E.70.3.7	stopWorker	270
E.70.3.8	reset	270
E.70.4	Member Data Documentation	270
E.70.4.1	_workers	270
E.71	BiometricEvaluation::Process::POSIXThreadWorkerController Class Reference	270
E.71.1	Detailed Description	271
E.71.2	Member Function Documentation	271
E.71.2.1	reset	271
E.71.2.2	isWorking	271
E.72	BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate Struct - Reference	272
E.72.1	Detailed Description	272
E.72.2	Constructor & Destructor Documentation	272
E.72.2.1	PrintPositionCoordinate	272
E.72.3	Member Data Documentation	273
E.72.3.1	fingerView	273
E.72.3.2	segment	273
E.72.3.3	coordinates	273
E.73	BiometricEvaluation::IO::Properties Class Reference	273
E.73.1	Detailed Description	274
E.73.2	Member Typedef Documentation	274
E.73.2.1	const_iterator	274
E.73.3	Constructor & Destructor Documentation	275
E.73.3.1	Properties	275
E.73.3.2	Properties	275
E.73.3.3	~Properties	275
E.73.4	Member Function Documentation	275
E.73.4.1	setProperty	275
E.73.4.2	setPropertyFromInteger	276
E.73.4.3	setPropertyFromDouble	276
E.73.4.4	removeProperty	276
E.73.4.5	getProperty	277
E.73.4.6	getPropertyAsInteger	277
E.73.4.7	getPropertyAsDouble	277
E.73.4.8	begin	278
E.73.4.9	end	278
E.73.4.10	getMode	278
E.73.4.11	initWithBuffer	278
E.73.4.12	initWithBuffer	279
E.74	BiometricEvaluation::IO::PropertiesFile Class Reference	279
E.74.1	Detailed Description	280
E.74.2	Constructor & Destructor Documentation	280
E.74.2.1	PropertiesFile	280
E.74.2.2	~PropertiesFile	280

E.74.3	Member Function Documentation	280
E.74.3.1	sync	281
E.74.3.2	changeName	281
E.75	BiometricEvaluation::Image::Raw Class Reference	281
E.75.1	Detailed Description	282
E.75.2	Member Function Documentation	282
E.75.2.1	getData	282
E.75.2.2	getRawData	282
E.75.2.3	getRawGrayscaleData	282
E.76	BiometricEvaluation::IO::RecordStore Class Reference	283
E.76.1	Detailed Description	285
E.76.2	Constructor & Destructor Documentation	286
E.76.2.1	RecordStore	286
E.76.2.2	RecordStore	286
E.76.3	Member Function Documentation	287
E.76.3.1	getName	287
E.76.3.2	getDescription	287
E.76.3.3	getCount	287
E.76.3.4	changeName	287
E.76.3.5	changeDescription	288
E.76.3.6	getSpaceUsed	288
E.76.3.7	sync	288
E.76.3.8	insert	289
E.76.3.9	insert	289
E.76.3.10	remove	289
E.76.3.11	read	290
E.76.3.12	read	290
E.76.3.13	replace	291
E.76.3.14	replace	291
E.76.3.15	length	291
E.76.3.16	flush	292
E.76.3.17	sequence	292
E.76.3.18	sequence	293
E.76.3.19	setCursorAtKey	293
E.76.3.20	openRecordStore	294
E.76.3.21	createRecordStore	294
E.76.3.22	removeRecordStore	295
E.76.3.23	mergeRecordStores	295
E.76.3.24	genKeySegName	296
E.76.3.25	setProperties	296
E.76.3.26	getProperties	296
E.76.4	Member Data Documentation	297
E.76.4.1	INVALIDKEYCHARS	297
E.76.4.2	KEY_SEGMENT_SEPARATOR	297
E.76.4.3	KEY_SEGMENT_START	297
E.76.4.4	CONTROLFILENAME	297
E.76.4.5	NAMEPROPERTY	297
E.76.4.6	DESCRIPTIONPROPERTY	297
E.76.4.7	COUNTPROPERTY	297
E.76.4.8	TYPEPROPERTY	297
E.76.4.9	BERKELEYDBTYPE	297

E.76.4.10 ARCHIVETYPE	298
E.76.4.11 FILETYPE	298
E.76.4.12 SQLITETYPE	298
E.76.4.13 COMPRESSEDTYPE	298
E.76.4.14 DEFAULTTYPE	298
E.76.4.15 RSREADONLYERROR	298
E.76.4.16 BE_RECSTORE_SEQ_START	298
E.76.4.17 BE_RECSTORE_SEQ_NEXT	298
E.77 BiometricEvaluation::View::AN2KView::RecordType Class Reference	298
E.77.1 Detailed Description	299
E.78 BiometricEvaluation::Image::Resolution Struct Reference	299
E.78.1 Detailed Description	299
E.78.2 Member Enumeration Documentation	299
E.78.2.1 Kind	299
E.78.3 Constructor & Destructor Documentation	300
E.78.3.1 Resolution	300
E.78.4 Member Data Documentation	300
E.78.4.1 xRes	300
E.78.4.2 yRes	300
E.78.4.3 units	300
E.79 BiometricEvaluation::Feature::RidgeCountExtractionMethod Class Reference	300
E.79.1 Detailed Description	301
E.80 BiometricEvaluation::Feature::RidgeCountItem Struct Reference	301
E.80.1 Detailed Description	301
E.81 BiometricEvaluation::Error::SignalManager Class Reference	301
E.81.1 Detailed Description	302
E.81.2 Constructor & Destructor Documentation	302
E.81.2.1 SignalManager	302
E.81.2.2 SignalManager	303
E.81.3 Member Function Documentation	303
E.81.3.1 setSignalSet	303
E.81.3.2 clearSignalSet	303
E.81.3.3 setDefaultSignalSet	303
E.81.3.4 sigHandled	303
E.81.3.5 start	304
E.81.3.6 stop	304
E.81.3.7 setSigHandled	304
E.81.3.8 clearSigHandled	304
E.81.4 Member Data Documentation	304
E.81.4.1 _canSigJump	304
E.81.4.2 _sigJumpBuf	304
E.82 BiometricEvaluation::Image::Size Struct Reference	305
E.82.1 Detailed Description	305
E.82.2 Constructor & Destructor Documentation	305
E.82.2.1 Size	305
E.82.3 Member Data Documentation	305
E.82.3.1 xSize	305
E.82.3.2 ySize	306
E.83 BiometricEvaluation::IO::SQLiteRecordStore Class Reference	306
E.83.1 Detailed Description	307
E.83.2 Member Function Documentation	307

E.83.2.1	changeName	307
E.83.2.2	changeDescription	307
E.83.2.3	getSpaceUsed	308
E.83.2.4	sync	308
E.83.2.5	insert	308
E.83.2.6	remove	309
E.83.2.7	read	309
E.83.2.8	replace	310
E.83.2.9	length	310
E.83.2.10	flush	310
E.83.2.11	sequence	311
E.83.2.12	setCursorAtKey	311
E.83.2.13	sqliteError	312
E.83.2.14	createStructure	312
E.83.2.15	validateKeyValueTable	312
E.83.2.16	createKeyValueTable	313
E.83.2.17	validateSchema	313
E.83.2.18	readSegments	313
E.83.2.19	cleanup	314
E.84	BiometricEvaluation::Process::Statistics Class Reference	314
E.84.1	Detailed Description	315
E.84.2	Constructor & Destructor Documentation	315
E.84.2.1	Statistics	315
E.84.2.2	Statistics	315
E.84.3	Member Function Documentation	315
E.84.3.1	getCPUTimes	315
E.84.3.2	getMemorySizes	316
E.84.3.3	getNumThreads	316
E.84.3.4	logStats	317
E.84.3.5	startAutoLogging	317
E.84.3.6	stopAutoLogging	317
E.84.3.7	callStatistics_logStats	318
E.85	BiometricEvaluation::Error::StrategyError Class Reference	318
E.85.1	Detailed Description	318
E.85.2	Constructor & Destructor Documentation	318
E.85.2.1	StrategyError	318
E.85.2.2	StrategyError	319
E.86	BiometricEvaluation::Time::Timer Class Reference	319
E.86.1	Detailed Description	319
E.86.2	Constructor & Destructor Documentation	319
E.86.2.1	Timer	319
E.86.3	Member Function Documentation	319
E.86.3.1	start	319
E.86.3.2	stop	320
E.86.3.3	elapsed	320
E.87	BiometricEvaluation::View::View Class Reference	320
E.87.1	Detailed Description	321
E.87.2	Member Function Documentation	321
E.87.2.1	getImage	321
E.87.2.2	getImageSize	321
E.87.2.3	getImageResolution	321

E.87.2.4	getImageDepth	321
E.87.2.5	getCompressionAlgorithm	322
E.87.2.6	getScanResolution	322
E.88	BiometricEvaluation::Time::Watchdog Class Reference	322
E.88.1	Detailed Description	323
E.88.2	Constructor & Destructor Documentation	323
E.88.2.1	Watchdog	323
E.88.3	Member Function Documentation	324
E.88.3.1	setInterval	324
E.88.3.2	start	324
E.88.3.3	stop	324
E.88.3.4	expired	325
E.88.3.5	setCanSigJump	325
E.88.3.6	clearCanSigJump	325
E.88.3.7	setExpired	325
E.88.3.8	clearExpired	325
E.88.4	Member Data Documentation	325
E.88.4.1	PROCESSTIME	325
E.88.4.2	REALTIME	325
E.89	BiometricEvaluation::Process::Worker Class Reference	325
E.89.1	Detailed Description	326
E.89.2	Member Function Documentation	326
E.89.2.1	workerMain	326
E.89.2.2	getParameter	327
E.89.2.3	getParameterAsDouble	327
E.89.2.4	getParameterAsInteger	327
E.89.2.5	getParameterAsString	328
E.89.2.6	setParameter	328
E.89.2.7	stop	328
E.89.2.8	stopRequested	328
E.90	BiometricEvaluation::Process::WorkerController Class Reference	329
E.90.1	Detailed Description	329
E.90.2	Constructor & Destructor Documentation	329
E.90.2.1	WorkerController	329
E.90.3	Member Function Documentation	330
E.90.3.1	setParameter	330
E.90.3.2	setParameterFromDouble	330
E.90.3.3	setParameterFromInteger	330
E.90.3.4	setParameterFromString	331
E.90.3.5	reset	331
E.90.3.6	isWorking	331
E.90.3.7	getWorker	331
E.90.4	Member Data Documentation	331
E.90.4.1	_worker	332
E.91	BiometricEvaluation::Image::WSQ Class Reference	332
E.91.1	Detailed Description	332
E.91.2	Member Function Documentation	332
E.91.2.1	getRawData	332
E.91.2.2	getRawGrayscaleData	333
E.91.2.3	isWSQ	333

Chapter 1

Introduction

This document describes the Biometric Evaluation Framework (BECCommon) and application programming interfaces (API) used to support the evaluation of biometric software within the NIST Image Group [12].

1.1 Rationale

When evaluating software in a “black box” fashion many aspects of program execution must be addressed, such as non-returning function calls, I/O errors, and other resource requirements. In addition, solutions to common problems should be portable across operating systems.

An evaluation consists of the testing of vendor-supplied software that implements certain biometric algorithms, such as fingerprint matching or face recognition. The NIST Image Group defines a test process and API for each evaluation. Vendors implement the API in their software, which is delivered to NIST as a software library, where common test driver is used to call the vendor library to perform the biometric operation. In order to support the common functionality used across all evaluations, such as logging, file input/output, etc., a common framework is used.

Even though the Biometric Evaluation Framework was written to support biometric software evaluations, much of the framework can be used for any general purpose programs where data storage and system interaction are needed. One goal of the BECommon is to reduce the low-level error processing (particularly with input and output) done directly by applications. The Biometric Evaluation Framework provides several abstractions that are useful to applications so they can focus on the task at hand.

This document describes the BECommon in two sections: Chapters containing descriptions of each package as well as code examples, and reference sections containing auto-generated API documentation.

The BECommon is a work-in-progress, and future development will occur in areas where the need arises for the testing programs of the NIST Image Group.

Chapter 2

Overview

The Biometric Evaluation Framework (BECCommon) is a set of C++[14] classes, error codes, and design patterns used to create a common environment to provide logging, data management, error handling, and other functionality that is needed for many applications used in the testing of biometric software. The goals of the framework include:

- Reduce the amount of I/O error handling implemented by applications.
- Provide standard interfaces for data management and logging;
- Remove the need for applications to handle low-level events from the operating system (signals, etc.);
- Provide services for timing the execution of code blocks;
- Allow applications to constrain the amount of processing time used by a block of code.

The experience of the NIST Image Group when running many software evaluations has led to the need of a common code for dealing with recurring software issues. One issue is the large amounts of data consumed, and created, by the software under test. Input data sets are typically biometric images, while output sets contain derived information. Both sets of data often contain millions of items, and storing each item as a file creates a tremendous burden on the file system. The *IO* package provides a solution to managing large amounts of records in a portable, efficient manner, as well as facilities for logging and maintaining runtime settings.

BECCommon is divided into several packages, each providing a set of related functionality, such as error handling and timing operations. The packages are an informal concept, mapped to formal C++ name spaces, e.g. *IO* and *Time*. A namespace contains classes, constants, and non-class functions that relate to concepts grouped in the namespace. All classes within BECCommon belong to the top-level *BiometricEvaluation* namespace.

Biometric image data is often supplied in a compressed format (e.g. WSQ, JPEG) and must be converted to a “raw” format. The *Image* package contains classes to represent compressed image data as an object, storing the image size and other attributes, in addition to the raw image.

Memory management issues are addressed by the *Memory* package. The use of classes and templates in this package can relieve applications of the need to directly manage memory for dynamically sized arrays, or call functions that are already provided to allocate and free C library objects.

While a program is running, it is often necessary to record certain statistics about the process, such as memory and processor usage. The *Process* package provides methods to obtain this information, as well as the capability to log to a file periodically, in an asynchronous manner.

In addition to its own statistics, a program may need to query some information about the environment under which it is running. The `System` package provides a count of CPUs, memory size, other system characteristics that an application can use to tailor its behavior.

Many aspects of software performance evaluation involve the use of timers. The `Time` package provides for the calculation of a time interval in a manner that is consistent across platforms, abstracting the underlying operating system's timing facility. Also, included is a "watchdog" facility, providing a solution to the problem of non-returning function calls. By using a watchdog timer, an application can abort a call to a function that doesn't return in the required interval.

The `Text` package provides a set of utility functions for operating on strings. The `digest` functions are of interest to those applications that must mask any information contained in a string before passing that information to another function. For example, often the biometric image file (or record) names contain information about the image, such as the finger position.

Error propagation and handling are addressed by the `Error` package. A set of exception objects are defined within this package, allowing for communication of error conditions out of the framework to the application, along with an explanatory string. Signal handling is related to error propagation in that when a process receives a signal, often it is due to software bug. Divide by zero, for example. The `Error` package provides for simple handling of the signal by the process.

Many packages in `BECommon` deal with biometric data record formats, including ANSI/NIST [?] records. In order to provide a general interface to several formats, `BECommon` represents the biometric data as derived from a source. For example, the `Finger` package contains classes that represent all information about a finger, including the source image and derived minutiae points. The `View` package combines the notions of a source image and derived information together into a single abstraction.

`BECommon` is designed to be used in a modular fashion, and it is possible to compile several packages independently. However, several packages do make use of other packages in the framework, and therefore, are less flexible in their reuse. However, `BECommon` is designed to reduce the intra-framework dependencies.

A set of test programs is included with the framework. These programs not only exercise the functions provided by the packages, but also can be used as example programs on how to use framework.

The chapters that follow this overview describe each package in detail, along with some code examples. The final set of chapters of this document contain the application programming interfaces for the types, methods, and classes that make up `BECommon`. However, the framework is under development, and other packages, classes, etc. will be added over time to address the needs of the NIST Image Group.

Chapter 3

Framework

The `Framework` package is used to retrieve information about the Biometric Evaluation Framework itself. Version numbers, the compiler used, and other information can be queried by applications. Versioning information is recorded in the `BECommon Makefile` and populated in the function implementation at compile-time.

Listing 3.1: Using the Framework API

```
1 #include <iostream>
2
3 #include <be_framework.h>
4
5 using namespace BiometricEvaluation;
6 using namespace std;
7
8 int
9 main(
10     int argc,
11     char* argv[])
12 {
13     cout << "Framework Version: ";
14     cout << Framework::getMajorVersion() << "." <<
15         Framework::getMinorVersion() << endl;
16     /* "Framework Version: 0.4" */
17
18     cout << "Compiler Used: ";
19     cout << Framework::getCompiler() << " v" <<
20         Framework::getCompilerVersion() << endl;
21     /* "Compiler Used: clang v3.0.0" */
22
23     cout << "Date/Time Compiled: ";
24     cout << Framework::getCompileDate() << " " <<
25         Framework::getCompileTime() << endl;
26     /* "Date/Time Compiled: Jan 24 2012 12:16:01" */
27
28     return (EXIT_SUCCESS);
29 }
```


Chapter 4

Memory

To assist applications with memory management, the `Memory` package provides classes to wrap C memory allocations, and other dynamically-sized objects.

4.1 AutoBuffer

The Biometric Evaluation Framework is designed to interoperate with existing C code that has its own memory management techniques, e.g. NIST Biometric Image Software [11]. In these cases, functions exist to allocate and free blocks of memory, and these calls must be made by the applications which use those libraries. To assist BECommon clients that use these existing libraries, the `AutoBuffer` class wraps the C memory management functions, guaranteeing the release of C objects when the `AutoBuffer` goes out of scope.

The `AutoBuffer` constructor takes three function pointers as parameters: one for C object construction, one for destruction, and a third, optional, function for copying the C object. If the latter is passed a `NULL`, the `AutoBuffer` and the underlying C object cannot be copied, and an exception will be thrown.

Listing 4.1 shows the use of `AutoBuffer` to wrap the memory allocation routines that are part of the NIST Biometric Image Software ANSI/NIST library.

Listing 4.1: Using the `AutoBuffer`

```
1 #include <be_memory_autobuffer.h>
2 #include <iostream>
3 extern "C" {
4     #include <an2k.h>
5 }
6
7 int
8 main(int argc, char* argv[]) {
9
10
11     /*
12      * alloc_ANSI_NIST(), free_ANSI_NIST(), and copy_ANSI_NIST()
13      * are functions in the NBIS AN2K library.
14      */
15     Memory::AutoBuffer<ANSI_NIST> an2k =
16         Memory::AutoBuffer<ANSI_NIST>(&alloc_ANSI_NIST,
17                                         &free_ANSI_NIST, &copy_ANSI_NIST);
```

```

18 |     if (read_ANSI_NIST(fp, an2k) != 0) {
19 |         cerr << "Could not read AN2K file." << endl;
20 |         return (EXIT_FAILURE);
21 |     }
22 |
23 |     for (int i = 1; i < an2k->num_records; i++) {
24 |         // process the ANSI/NIST record ...
25 |     }
26 | }

```

4.2 AutoArray

At its simplest level, `AutoArray` is a C-style array with numerous convenience methods, such as being able to query the number of elements. C++ iterators can be used over the contents of the array. The array can be resized without the need to create a new object. C++ operator overloading allows `AutoArray` objects to be passed to C-style functions that expect pointers to `AutoArray`'s template type.

`AutoArray` is used extensively in `BECommon` to help eliminate mistakes when manually allocating memory. The `AutoArray` constructor will allocate needed memory using `new` and the destructor will `delete` it. This ensures that any allocated memory will be appropriately freed when the `AutoArray` goes out of scope. Copy constructors and methods as well as the assignment operator all correctly manage memory so the client does not have to. Several objects in `BECommon` return `AutoArray` objects to assist clients in proper memory management.

A common use of `AutoArray` is to deal with records sequenced from a `RecordStore`. Listing 4.2 demonstrates this. Notice the omission of memory management statements – they are completely unnecessary.

Listing 4.2: Using `AutoArrays` with `RecordStores`

```

1 | #include <be_io_dbrecstore.h>
2 | #include <be_memory_autoarray.h>
3 |
4 | #include <iostream>
5 |
6 | using namespace BiometricEvaluation;
7 |
8 | int
9 | main(
10 |     int argc,
11 |     char *argv[])
12 | {
13 |     IO::DBRecordStore rs("db_recstore", ".", IO::READONLY);
14 |
15 |     uint64_t value_size = 0;
16 |     string key("");
17 |     Memory::AutoArray<uint8_t> value;
18 |     for (bool stop = false; stop == false; ) {
19 |         try {
20 |             // Non-destructively resize the AutoArray to hold
21 |             // the next record.
22 |             value.resize(rs.sequence(key, NULL));
23 |
24 |             // Read the record into the AutoArray (treats the
25 |             // AutoArray as a pointer).

```

```

26         rs.read(key, value);
27
28         // Do something with value.
29         std::cout << "Key " << key << " has a value of " <<
30             value.size() << " bytes" << std::endl;
31     } catch (Error::ObjectDoesNotExist) {
32         stop = true;
33     }
34 }
35
36 return (0);
37 }

```

AutoArray is adapted from "c_array" [14, 496].

4.3 IndexedBuffer

Many applications have a need to read items from a data record and take action based on the value of the item read. For example, when reading a biometric data record, the number of finger minutiae points in the record is indicated by a value in the record header. Furthermore, the record format may be of a different endianness than the application's host platform.

The `IndexedBuffer` class is used to access data from a buffer in fixed-size amounts in sequence. Objects of this class maintain an index into the buffer as internal state and reads out of the buffer, when using certain methods, adjust the index. In addition, standard subscript access can be done on the buffer (reads and writes) without affecting the index. The basic element type is an unsigned eight-bit value. The `IndexedBuffer` object can be created to either manage the buffer memory directly, or to "wrap" an existing buffer.

Methods to retrieve elements from the buffer are defined in the class's interface. These functions are used to retrieve 8/16/32/64-bit values while moving the internal index. Several functions are also provided to take into account the endianness of the underlying data.

Listing 4.3 shows how an application can read a data record in big-endian format.

Listing 4.3: Using the `IndexedBuffer`

```

1 #include <be_memory_autoarray.h>
2 #include <be_memory_indexedbuffer.h>
3
4 int
5 main(int argc, char* argv[]) {
6
7     uint64_t size = IO::Utility::getFileSize("BiometricRecord");
8     FILE *fp = std::fopen("BiometricRecord", "rb");
9     Memory::IndexedBuffer iBuf(size);
10    fread(iBuf, 1, size, fp);
11    fclose(fp);
12    Memory::IndexedBuffer iBuf(recordData, recordData.size());
13
14    uint32_t lval;
15    uint16_t sval;
16
17    /*
18     * Record is big-endian:
19     * -----

```

```
20      * | NAME | LENGTH | ID |      ...      |
21      * -----
22      *      4      4      2
23      */
24
25      /* Read a 4-byte C string */
26      lval = iBuf.scanU32Val();          /* Format ID */
27      char *cptr = (char *)&lval;
28      string s(cptr);
29
30      /* Read a 4-byte length */
31      lval = iBuf.scanBeU32Val()
32
33      /* Read a 2-byte ID */
34      sval = iBuf.scanBeU16Val();
35 }
```


Chapter 5

Error Handling

Within the Biometric Evaluation Framework, Error handling has two aspects: One for communicating error conditions out of the framework and back to applications; the other for handling error signals from the environment and operating system. Classes and other code to implement error processing are described in this chapter.

5.1 Biometric Evaluation Exceptions

The Biometric Evaluation Framework contains a set of classes used to report errors to applications. Objects of these class types are thrown and contain descriptive information as to the nature of the error. Applications must handle the errors in a manner that makes sense for the application.

Applications should catch objects of the type specified in the API for the class being called. The type of object caught indicates the nature of the error that occurred, while the string stored within that object provides more information on the error.

Listing [6.2 on page 17](#) shows an example of exception handling when using the logging classes described in Section [6.3 on page 17](#).

5.2 Signal Handling

When the application process executes in a POSIX environment, signals to the process can be generated by the operating system. In many cases, if the signal is not handled by the process, execution terminates. Because the Biometric Evaluation Framework was designed to be used with software libraries for which no source code is available, changes to the code in these libraries cannot be made, and any faults in that code cannot be fixed. A common problem is that a function in the “black box” library dereferences a bad pointer, resulting in a segmentation violation signal being sent by the operating system.

To prevent termination of the application process, signal handling must be installed. The Biometric Evaluation Framework provides a class, `SignalManager`, to simplify the installation of a signal handler in order to allow the program to continue running. For example, when extracting a fingerprint minutia template from an image, often the library call will fault on a certain image. By using the `SignalManager`, the application can log that fault, and continue on to the next image.

Signal handling in a POSIX environment covers the bare essentials, and one of two actions is usually taken.

The signal can be handled and processing continues at the location the signal was generated. The second action is that, in addition to signal handling, the process continues from a different location. It is the second action that is implemented by the `SignalManager` class. The rationale for this type of signal handling is so the call to the faulting function can be aborted, but the caller can detect that the signal was handled and take action, usually by logging the fault.

By default, the `SignalManager` class installs a handler for the `SIGSEGV` and `SIGBUS` signals. However, other signals can be handled as desired.

One restriction on the use of `SignalManager` is that the POSIX calls for signal management (`signal(3)`, `sigaction(2)`, etc.) cannot be invoked inside of the signal handler block.

The example in Listing 5.1 shows application use of the `SignalManager` class.

Listing 5.1: Using the `SignalManager`

```

1 #include <be_error_signal_manager.h>
2 using namespace BiometricEvaluation;
3
4 int main(int argc, char *argv[])
5 {
6     Error::SignalManager *sigmgr = new Error::SignalManager();
7
8     BEGIN_SIGNAL_BLOCK(sigmgr, sigblock1);
9     // code that may result in signal generation
10    END_SIGNAL_BLOCK(sigmgr, sigblock1);
11    if (sigmgr->sigHandled()) {
12        // log the event, etc.
13    }
14 }
```

Within the `SignalManager` header file, two macros are defined: `BEGIN_SIGNAL_BLOCK()` and `END_SIGNAL_BLOCK()`, each taking the `SignalManager` object and label as parameters. The label must be unique for each signal block. These macros insert the jump buffer into the code, which is the location where the signal handler will jump to after handling the signal. The use of these macros greatly simplifies signal handling for the application, and it is recommended that applications use these macros instead of directly invoking the methods of the `SignalManager` class, except for changing the set of handled signals.

If a signal does occur, process control jumps to the end of the signal block, and the `sigHandled()` method of the signal manager can be called. The application may need to have the same statements inside the `sigHandled()` check as those outside of the signal handling block. For example, if a file needs to be closed before the end of the block, the same call to the close function must be made within the `sigHandled()` check. Careful application design can reduce the amount of code replication, however.

Listing 5.2 shows how an application can indicate what signals to handle. In this example, only the `SIGUSR1` signal would be handled.

Listing 5.2: Specifying Signals to the `SignalManager`

```

1 #include <be_error_signal_manager.h>
2 using namespace BiometricEvaluation;
3
4 int main(int argc, char *argv[])
5 {
6     Error::SignalManager *sigmgr = new Error::SignalManager();
7
8     sigset_t sigset;
```

```
9      sigemptyset(&sigset);
10     sigaddset(&sigset, SIGUSR1);
11     sigmgr->setSignalSet(sigset);
12
13     FILE *fp = fopen( ... );
14     BEGIN_SIGNAL_BLOCK(sigmgr, sigblock2);
15         // code that may result in signal generation
16         fclose(fp);
17     END_SIGNAL_BLOCK(asigmgr, sigblock2);
18     if (sigmgr->sigHandled()) {
19         cout << "SIGUSR1 occurred." << endl;
20         fclose(fp);
21     }
22 }
```


Chapter 6

Input/Output

The `IO` package is used by applications for the common types of input and output: managing stores of data, log files, and individual file management. The goal of using the `IO` API is to relieve applications of the need to manage low-level I/O operations such as file opening, writing, and error handling. Furthermore, by using the classes defined in `IO`, the actual storage mechanism used for data can be managed efficiently and placed in a consistent location for all applications.

Many classes manage persistent storage within the file system, taking care of file open and close operations, as well as error handling. When errors do occur, exceptions are thrown, which then must be handled by the application.

6.1 Utility

The `IO::Utility` namespace provides functions that are used to manipulate the file system and other low-level mechanisms. These functions can be used by applications in addition to being used by other classes within the Biometric Evaluation framework. The functions in this package are used to directly manipulate objects in the POSIX file system, or to check whether a file object exists.

6.2 Record Management

The `IO::RecordStore` class provides an abstraction for performing record-oriented input and output to an underlying storage system. Each implementation of the `RecordStore` provides a self-contained entity to manage data on behalf of the application in a reliable, efficient manner.

Many biometric evaluations generate thousands of files in the form of processed images and biometric templates, in addition to consuming large numbers of files as input. In many file systems, managing large numbers of files is not efficient, and leads to longer run times as well as difficulty in backing up and processing these files outside of the actual evaluation.

The `RecordStore` abstraction de-couples the application from the underlying storage, enabling the implementation of different strategies for data management. One simple strategy is to store each record into a separate file, reproducing what has typically been done in the evaluation software itself. Archive files and small databases are other implementation strategies that have been used.

Use of the `RecordStore` abstraction allows applications to switch storage strategy by changing a few lines

of code. Furthermore, error handling is consistent for all strategies by the use of common exceptions.

RecordStores provide no semantic meaning to the nature of the data that passes through the store. Each record is an opaque object, given to the store as a pointer and data length, and is associated with a string the which is the key. Keys must be unique and are associated with a single record. Attempts to insert multiple records with the same key result in an exception being thrown.

Listing 6.1 illustrates the use of a database RecordStore within an application.

Listing 6.1: Using a RecordStore

```

1 #include <iostream>
2 #include <be_io_dbrecstore.h>
3 int
4 main(int argc, char* argv[]) {
5
6     IO::DBRecordStore *rs;
7     try {
8         rs = new IO::DBRecordStore("myRecords", "My Record Store", "");
9     } catch (Error::Exception& e) {
10         cout << "Caught " << e.getInfo() << endl;
11         return (EXIT_FAILURE);
12     }
13     auto_ptr<IO::DBRecordStore> ars(rs);
14
15     try {
16         uint8_t *theData;
17
18         theData = getSomeData();
19         ars->insert("key1", theData);
20
21         theData = getSomeData();
22         ars->insert("key2", theData);
23
24     } catch (Error::Exception& e) {
25         cout << "Caught " << e.getInfo() << endl;
26         return (EXIT_FAILURE);
27     }
28
29     // Some more processing where new data for a key comes in ...
30     theData = getSomeData();
31     ars->replace("key1", theData);
32
33     // Obtain the data for all keys ...
34     string theKey;
35     while (true) {
36         uint64_t len = rs->sequence(theKey, theData);
37         cout << "Read data for key " << theKey << " of length " << len << endl;
38     }
39     // The data for the key is no longer needed ...
40     ars->remove("key1");
41 }

```

6.3 Logging

Many applications are required to log information during their processing. In particular, the evaluation test drivers often create a log record for each call to the software under test. There is a need for the log entries to be consistent, yet any logging facility must be flexible in accepting the type of data that is to be written to the log file.

The logging classes in the `IO` package provide a straight-forward method for applications to record their progress without the need to manage the low-level output details. There are two classes, `IO::LogCabinet` and `IO::LogSheet` that are used to perform consistent logging of information by applications. A `LogCabinet` contains a set of `LogSheets`.

A `LogSheet` is an output stream (subclass of `std::ostream`), and therefore can handle built-in types and any class that supports streaming. The example code in Listing 6.2 shows how an application can use a `LogSheet`, contained within a `LogCabinet`, to record operational information.

Log sheets are simple text files, with each entry numbered by the `LogSheet` class when written to the file. The description of the sheet is placed at the top of the file during construction of the `LogSheet` object. A call to the `newEntry()` method commits the current entry to the log file, and resets the write position to the beginning of the entry buffer.

In addition to streaming by using the `LogSheet::«` operator, applications can directly commit an entry to the log file by calling the `write()` method, thereby not disrupting the entry that is being formed. After an entry is committed, the entry number is automatically incremented.

The example in Listing 6.2 shows application use of the logging facility.

Listing 6.2: Using a `LogSheet` within a `LogCabinet`

```

1 #include <be_io_logcabinet.h>
2 using namespace BiometricEvaluation;
3 using namespace BiometricEvaluation::IO;
4
5 LogCabinet *lc;
6 try {
7     lc = new LogCabinet(lcname, "A Log Cabinet", "");
8 } catch (Error::ObjectExists &e) {
9     cout << "The Log Cabinet already exists." << endl;
10    return (-1);
11 } catch (Error::StrategyError& e) {
12    cout << "Caught " << e.getInfo() << endl;
13    return (-1);
14 }
15 auto_ptr<LogCabinet> alc(lc);
16 try {
17     ls = alc->newLogSheet(lsname, "Log Sheet in Cabinet");
18 } catch (Error::ObjectExists &e) {
19     cout << "The Log Sheet already exists." << endl;
20     return (-1);
21 } catch (Error::StrategyError& e) {
22     cout << "Caught " << e.getInfo() << endl;
23     return (-1);
24 }
25 ls->setAutoSync(true); // Force write of every entry when finished
26 int i = ...
27 *ls << "Adding an integer value " << i << " to the log." << endl;
28 ls->newEntry();        // Forces the write of the current entry

```

```

29| .....
30| delete ls;
31| return;                // The LogCabinet is destructed by the auto_ptr

```

6.4 Properties

The `Properties` class is used to store simple key-value string pairs, with the option to save to a file. Applications can use a `Properties` object to manage runtime settings that are persistent across invocations, or to simply store some settings in memory only.

Listing 6.3: Using a `Properties` Object

```

1| IO::Properties *props;
2| string fname = "test.prop";
3| try {
4|     props = new IO::Properties(fname);
5| } catch (Error::StrategyError &e) {
6|     cerr << "Caught " << e.getInfo() << endl;
7|     return;
8| } catch (Error::FileError& e) {
9|     cerr << "A file error occurred: " << e.getInfo() << endl;
10|    return;
11| }
12| props->setProperty("foo", "bar");
13| props->setProperty("theAnswer", "42");
14| :
15| :
16| :
17| try {
18|     int64_t theAnswer = props->getProperty("theAnswer");
19|     cout << "The answer is " << theAnswer << endl;
20| } catch (Error::ObjectDoesNotExist &e) {
21|     cerr << "The answer is elusive." << endl;
22|     return;
23| }
24| string fooProp = props->getProperty("foo");
25| cout << "Foo is set to " << fooProp << endl;
26| :
27| :
28| :
29| try {
30|     props->removeProperty("foo");
31| } catch (Error::ObjectDoesNotExist &e) {
32|     cerr << "Failed to remove property." << endl;
33| }

```

6.5 Compressor

Support for data compression and decompression can be found in the Biometric Evaluation Framework through the `Compressor` class hierarchy. `Compressor` is an abstract base class defining several pure-virtual methods for compression and decompression of buffers and files. Derived classes implement these methods and can

be instantiated through the factory method in the base class. As such, children should also be enumerated within `Compressor::Kind`. The Biometric Evaluation Framework comes with an example, GZIP, which compresses and decompresses the gzip format through interaction with `zlib` [4].

Listing 6.4: Using a Compressor Object

```

1 tr1::shared_ptr<IO::Compressor> compressor;
2 Memory::uint8Array compressedBuffer, largeBuffer = /* ... */;
3 try {
4     compressor = IO::Compressor::createCompressor(Compressor::Kind::GZIP);
5     /* Overloaded for all combination of buffer and file */
6     compressor->compress("largeInputFile", "compressedOutputFile");
7     compressedBuffer = compressor->compress(largeBuffer);
8 } catch (Error::Exception &e) {
9     cerr << "Could not compress (" << e.getInfo() << ')' << endl;
10 }

```

Different Compressors may be able to respond to options that tune their operations. These options (and approved values) should be well-documented in the child class, however, a no-argument constructor of a child Compressor should automatically set any required options to default values. Setting and retrieving these options is very similar to interacting with a Properties object (see Section 6.4 on the facing page).

Listing 6.5: Setting Compressor Options

```

1 tr1::shared_ptr<IO::Compressor> compressor =
2     IO::Compressor::createCompressor(Compressor::Kind::GZIP);
3
4 /* A large GZIP chunk size can speed operations on systems with copious RAM */
5 compressor->setOption(IO::GZIP::CHUNK_SIZE, 32768);

```


Chapter 7

Time and Timing

The `Time` package within the Biometric Evaluation Framework provides a set of classes for performing timing-related operations, such as elapsed time and limiting execution time.

7.1 Elapsed Time

The `Timer` class provides applications a method to determine how long a block of code takes to execute. On many systems (e.g. Linux) the timer resolution is in microseconds.

Listing 7.1 shows how an application can use a `Timer` object to limit obtain the amount of time used for the execution of a block of code.

Listing 7.1: Using the `Timer`

```
1 #include <be_time_timer.h>
2
3 int main(int argc, char *argv[])
4 {
5     Time::Timer timer = new Time::Timer();
6
7     try {
8         atimer->start();
9         // do something useful, or not
10        atimer->stop();
11        cout << "Elapsed time: " << atimer->elapsed() << endl;
12    } catch (Error::StrategyError &e) {
13        cout << "Failed to create timer." << endl;
14    }
15 }
```

7.2 Limiting Execution Time

The `Watchdog` class allows applications to control the amount of time that a block of code has to execute. The time can be *real* (i.e. “wall”) time, or *process* time (not available on Windows). One typical usage for a

Watchdog timer is when a call is made to a function that may never return, due to problems processing an input biometric image.

Watchdog timers can be used in conjunction with `SignalManager` in order to both limit the processing time of a call, and handle all signals generated as a result of that call. See 5.2 for information on the `SignalManager` class.

One restriction on the use of Watchdog is that the POSIX calls for signal management (`signal(3)`, `sigaction(2)`, etc.) cannot be invoked inside of the WATCHDOG block. This restriction includes calls to `sleep(3)` because it is based on signal handling as well.

Listing 7.2 shows how an application can use a Watchdog object to limit the amount of process time for a block of code.

Listing 7.2: Using the Watchdog

```

1 #include <be_time_watchdog.h>
2 int main(int argc, char *argv[])
3
4     Time::Watchdog theDog = new Time::Watchdog(Time::Watchdog::PROCESSTIME);
5     theDog->setInterval(300);    // 300 microseconds
6
7     Time::Timer timer;
8
9     BEGIN_WATCHDOG_BLOCK(theDog, watchdogblock1);
10        timer.start();
11        // Do something that may take more than 300 usecs
12        timer.stop();
13        cout << "Total time was " << timer.elapsed() << endl;
14    END_WATCHDOG_BLOCK(theDog, watchdogblock1);
15    if (theDog->expired()) {
16        timer.stop();
17        cerr << "That took too long." << endl;
18    }
19 {
20 }
```

Within the Watchdog header file, two macros are defined: `BEGIN_WATCHDOG_BLOCK()` and `END_WATCHDOG_BLOCK()`, each taking the Watchdog object and label as parameters. The label must be unique for each WATCHDOG block. The use of these macros greatly simplifies Watchdog timers for the application, and it is recommended that applications use these macros instead of directly invoking the methods of the Watchdog class, except for setting the timeout value.

Any processing that is normally done at the end of the WATCHDOG block must also be done within the `expired()` check due to the fact that process control jumps to the end of the WATCHDOG block in the event of a timeout. A typical example is the use of the `Timer` object inside a WATCHDOG block, as the example in Listing 7.2 shows. In most cases, however, careful application design can remove the need for duplicate code. In the example, placing the `Timer start()/stop()` calls outside of the WATCHDOG block simplifies the coding, although the small amount of time for the WATCHDOG setup and tear down would be included in the time.

Chapter 8

Process Information

The `Process` package is a set of APIs used to gather information on a process, limit the capabilities of a process, and create manage processes.

8.1 Process Statistics

When a application is running, there is a need to obtain information of the process executing that application. The `Process` API can be used by the application itself to gather statistics related to the current amount of memory being used, the number of threads, and other items. Biometric evaluation test drivers are linked against a third party library, and therefore, the application writer does not control the thread count or memory usage for much of the processing. Listing 8.1 shows how an application can use the `Statistics` API.

Listing 8.1: Gathering Process Statistics

```
1 #include <be_error_exception.h>
2 #include <be_process_statistics.h>
3 using namespace BiometricEvaluation;
4
5 int main(int argc, char *argv[])
6 {
7     Process::Statistics stats;
8     uint64_t userstart, userend;
9     uint64_t systemstart, systemend;
10    uint64_t diff;
11    try {
12        stats.getCPUTimes(&userstart, &systemstart);
13
14        // Do some long processing....
15
16        stats.getCPUTimes(&userend, &systemend);
17        diff = userend - userstart;
18        cout << "User time elapsed is " << diff << endl;
19        diff = systemend - systemstart;
20        cout << "System time elapsed is " << diff << endl;
21    } catch (Error::Exception) {
22        cout << "Caught " << e.getInfo() << endl;
23    }
```

24 |
25 | }

In addition to using the `Process` API to gather statistics to be returned from the function call, the API provides a means to have a “standard” set of statistics logged either synchronously or asynchronously to a `LogSheet` (See Section 6.3 on page 17) contained within a `LogCabinet`. Applications can start and stop logging at will to this `LogSheet`. Post-mortem analysis can then be done on the entries in the `LogSheet`. Listing 8.2 shows the use of logging.

The `LogSheet` will have a file name constructed from the process name (i.e. the application executable) and the process ID. An example `LogSheet` contains this information at the start:

```
Description: Statistics for test_be_process_statistics (PID 28370)
# Entry Uptime Systeime RSS VMSize VMPeak VMData VMStack Threads
E0000000001 728889 6998 1788 57472 62612 31020 84 1
E0000000002 1300802 6998 1792 57472 62612 31020 84 1
```

The `Statistics` object creates the `LogSheet` with an appropriate description and comment entry with column headers. Each gathering of the statistics results in a single log entry.

Listing 8.2: Logging Process Statistics

```
1 #include <be_error_exception.h>
2 #include <be_io_logcabinet.h>
3 #include <be_process_statistics.h>
4 using namespace BiometricEvaluation;
5
6 int main(int argc, char *argv[])
7 {
8     IO::LogCabinet lc("statLogCabinet", "Cabinet for Statistics", "");
9
10    Process::Statistics *logstats;
11    try {
12        logstats = new Process::Statistics(&lc);
13    } catch (Error::Exception &e) {
14        cout << "Caught " << e.getInfo() << endl;
15        return (EXIT_FAILURE);
16    }
17    try {
18        while (some_processing_to_do) {
19            // Do the work
20            // Synchronously log after the work is done.
21            logstats->logStats();
22        }
23    } catch (Error::Exception &e) {
24        cout << "Caught " << e.getInfo() << endl;
25        delete logstats;
26        return (EXIT_FAILURE);
27    }
28
29    // Set up asynchronous logging, every second
30    try {
31        logstats->startAutoLogging(1);
32    } catch (Error::ObjectExists &e) {
33        cout << "Caught " << e.getInfo() << endl;
```

```

34         delete logstats;
35         return (EXIT_FAILURE);
36     }
37
38     // Do some other work
39
40     // Stop logging
41     logstats->stopAutoLogging();
42     delete logstats;
43 }

```

8.2 Process Management

During a biometric evaluation or other long-running CPU-bound task, it's beneficial to make efficient use of all the hardware available on the system. If your application is running on a multi-core machine, why not make use of more than one core? BECommon aims to simplify this by abstracting the usage of `fork(2)` and `libpthread` to run multiple instances of the same function simultaneously.

8.2.1 Manager

There are three class hierarchies involved in the abstraction. The `Manager` classes control the technique of process manipulation that will be used. BECommon provides two example abstractions: `ForkManager` and `POSIXThreadManager`. When using `ForkManager`, new processes will be created with `fork(2)`, with mediated access to these new processes through the `Manager`. Likewise, `POSIXThreadManager` creates new POSIX threads. Because both of these classes inherit from `Manager`, it is as trivial as changing the `Manager` object type to change how the workload is parallelized.

8.2.2 Worker

In the application using a `Manager`, a `Worker` subclass must be implemented. An example `Worker` is shown in Listing 8.3. The entry-point for a `Worker` is the `workerMain()` method, which must be implemented by the client application. Although `workerMain()` takes no arguments, data may be transmitted into the object through `WorkerController`'s (8.2.3) `setParameter()` method. Within the `Worker` instance, the parameters are then retrieved with `getParameter()` when provided with the unique parameter name.

A responsible `Worker` performs its operations as fast as it can, however, at any given time, the `Manager` may ask the `Worker` to stop. It then becomes the *responsibility of the Worker* to stop as soon as possible. The `Worker` is notified of the stop request through its `stopRequested()` method. Note that the `Manager` does **not** force the `Worker` to stop, though prolonged work or cleanup in the `Worker` would likely produce undesired results in the client application. As such, a responsible `Worker` checkpoints itself to prepare for premature stops requested by the `Manager`. While it is important for `Workers` to stop as soon as possible after the request is received, it is also important not to leave work in an unsynchronized state. In Listing 8.3, notice how the `Employee` must continue the interaction with the `Customer` before a stop request is handled, even if the `Employee`'s shift has ended. Leaving the method before the `Customer`'s order has been delivered would leave the `Customer` object in an unsafe state (hungry).

Listing 8.3: A Responsible Worker Implementation

```

1 #include <cstdlib>
2 #include <tr1/memory>

```

```

3 #include <queue>
4
5 #include <restaurant.h>
6
7 #include <be_process_forkmanager.h>
8
9 using namespace std;
10 using namespace BiometricEvaluation;
11 using namespace Restaurant;
12
13 class ResponsibleEmployeeTask : public Process::Worker
14 {
15 public:
16     int32_t
17     workerMain()
18     {
19         int32_t status = EXIT_FAILURE;
20
21         /* Retrieve objects assigned to this Task */
22         tr1::shared_ptr<Employee> employee =
23             tr1::static_pointer_cast<Employee>(
24                 this->getParameter("employee"));
25         tr1::shared_ptr< queue<Customer*> > customers =
26             tr1::static_pointer_cast< queue<Customer*> >(
27                 this->getParameter("customers"))
28
29         employee->clockIn();
30
31         Customer *customer;
32         /* Checkpoint after each customer */
33         while (this->stopRequested() == false ||
34             employee->isShiftOver() == false) {
35             customer = customers->front();
36
37             if (customer != NULL) {
38                 employee->takeOrder(customer);
39                 employee->cookFood(customer);
40                 employee->deliverOrder(customer);
41
42                 customers->pop();
43             }
44         }
45
46         employee->settleCashDrawer();
47         employee->clockOut();
48
49         status = EXIT_SUCCESS;
50         return (status);
51     }
52     ~ResponsibleEmployeeTask() {}
53 };

```

After a Manager starts its Workers, the Manager has the option of waiting until all Workers exit worker Main() before continuing code execution. If not waiting, there are several methods the Manager can perform to keep track of the status of the Workers. Even if not waiting for Workers to return, a responsible Manager will wait a reasonable amount of time for Workers to return before application termination. An example

of this reasonable waiting period can be seen in Listing 8.4.

8.2.3 WorkerController

The final piece of the process management puzzle is the WorkerController hierarchy. This class decorates and mediates communication between the Manager and the Worker. WorkerController objects may only be instantiated by a Manager object. All communications to the Worker (e.g. `isWorking()`) should be delegated through the WorkerController. If defining a new Manager, note that the WorkerController may seem unnecessary for the parallelization technique being employed. It's true that some parallelization techniques may not require this "middle-man" approach, but others do. Do not be concerned if a WorkerController implementation ends up being nothing more than a "pass-thru" to the Worker.

Listing 8.4 is a continuation of Listing 8.3 on page 25 demonstrating the use of Managers and Worker Controllers.

Listing 8.4: Using Managers and WorkerControllers

```

1 int
2 main(
3     int argc,
4     char *argv[])
5 {
6     static const uint32_t numEmployees = 3;
7     int status = EXIT_FAILURE;
8
9     tr1::shared_ptr<Process::Manager> shiftLeader(new Process::ForkManager);
10    queue<Customer*> *customers = new queue<Customer*>();
11
12    /* Create Employees (Workers/WorkerControllers) */
13    tr1::shared_ptr<Process::WorkerController> employees[numEmployees];
14    for (uint32_t i = 0; i < numEmployees; i++) {
15        employees[i] = shiftLeader->addWorker(
16            tr1::shared_ptr<ResponsibleEmployeeTask>(
17                new ResponsibleEmployeeTask()));
18
19        /* Assign employees to each Task */
20        employees[i]->setParameter("employee",
21            tr1::shared_ptr<Employee>(new Employee()));
22        employees[i]->setParameter("customers",
23            tr1::shared_ptr<queue<Customer*>>(customers));
24    }
25
26    /* Employees start serving customers while shift leader manages */
27    shiftLeader->startWorkers(false);
28
29    /* Customers enter the queue ... */
30    queue<Restaurant::AdministrativeTasks> adminTasks;
31    adminTasks.push("Inventory");
32    adminTasks.push("Customer Complaints");
33    adminTasks.push("Clean Dining Room");
34
35    while (shiftLeader->getNumActiveWorkers() != 0) {
36        shiftLeader->doTask(adminTasks.front());
37        adminTasks.pop();
38    }

```

```
39|
40|     /* ...end of the day */
41|     for (uint32_t i = 0; i < numEmployees; i++)
42|         if (employees[i]->isWorking()
43|             shiftLeader->stopWorker(employees[i]);
44|
45|     /*
46|      * Wait a reasonable amount of time before locking up for the night
47|      * (in this case, indefinitely).
48|     */
49|     while (shiftLeader->getNumActiveWorkers() > 0)
50|         sleep(1);
51|
52|     shiftLeader->armAlarmAndExit();
53|
54|     status = EXIT_SUCCESS;
55|     return (status);
56| }
```

Chapter 9

System

The `System` package provides a set of functions in the that return information about the hardware and operating system. This information can be used by applications to determine the amount of real memory, number of central processing units, or current load average. This information can be used to dynamically tailor the application behavior, or simply to provide additional information in a runtime log.

Listing 9.1 shows how an application can spawn several child processes based on the number of CPUs and memory available. Note that this information may not be available on all platforms, and therefore, the application must be prepared to handle that situation.

Listing 9.1: Using the `System` CPU Count Information

```
1 #include <iostream>
2 #include <be_system.h>
3
4 using namespace BiometricEvaluation;
5
6 int
7 main(int argc, char* argv[]) {
8
9     // perform some application setup ...
10
11     uint32_t cpuCount;
12     uint64_t memSize, vmSize;
13     try {
14         cpuCount = System::getCPUCount();
15         cpuCount--; // subtract one CPU for the parent process
16         memSize = System::getRealMemorySize();
17         Process::Statistics::getMemorySizes(NULL, &vmSize, NULL, NULL, NULL);
18         memSize -= vmSize; // subtract off memory used by parent
19
20         // Give each child a fraction of the memory
21         spawnChildren(cpuCount, memSize / cpuCount);
22     } catch (Error::NotImplemented) {
23         cout << "Running a single process only." << endl;
24     }
25
26     // processing done by parent ...
27 }
```


Chapter 10

Image

The `Image` package maintains the classes and other information related to images and image processing. Within the Biometric Evaluation Framework, many classes refer to images, such as when dealing with fingerprint data. Many biometric data standards supply the actual image encoded in one of several standard formats. Applications can retrieve the image as stored in the record, or decompressed by the `Image` class into a “raw” format. Therefore, within the `BECommon`, several of the common compression formats are supported, removing the need for applications to decompress the image directly, while maintaining access to the as-recorded image format.

10.1 The Image Namespace

The `Image` namespace contains several data types used to represent aspects of an image. The types defined are chiefly used to retrieve common information from images stored in an `Image` class (section 10.2). Data types in the `Image` namespace do not perform any translation of scale units or sizing, as each set of attributes is copied directly from the image data itself when possible.

The same applies to images encapsulated in biometric records. Although some biometric records have fields for image attributes like dimensions and resolution, the corresponding fields of an `Image` class are **not** populated with their contents. The `Image` namespace data types *are* used outside of the namespace, such as in finger views, to retrieve image attributes stored as part of the biometric record. Applications can compare those values against the values within the `Image` object, as in most cases those values are taken directly from the underlying image data. See Chapter 14 on page 41 for more information on image-based biometric records.

The `Image` namespace contains all of the `Image` classes that are used to represent an image. These classes are described in the following sections.

10.2 The Image Class

The `Image` class is an abstract base class that defines a set of minimum functionality for all supported image formats. Once an `Image` has been constructed, it may not be modified. For any supported image format, the following information is required to be accessible:

- Original binary data
- Compression algorithm

- Decompressed (“raw”) format binary data (grayscale, full color)
- Depth
- Dimensions (width, height)
- Resolution (horizontal, vertical)

A rudimentary implementation of generating a grayscale image is provided by the `Image` class in `getRawGrayscaleData()`. This implementation calculates the luminance value Y (of $YCbCr$) for each pixel of a color image. The resulting image always uses 8-bits to represent a pixel, but can return a raw image using 2 gray levels (1-bit) or 256 gray levels (8-bit). The 1-bit algorithm quantizes to black when the 8-bit color value is ≤ 127 . Image subclasses may override and implement their own grayscale conversion methods.

Also of interest in the `Image` class is `valueInColorspace()`, a static function to convert color values between bit depths.

10.3 Raw Image

The `RawImage` class represents a decompressed image, or an image where `getRawData()` would return the exact same data as `getData()`. `RawImage` has no special implementation or additional methods.

10.4 JPEG

The `JPEG` class represents an image encoded according to the JPEG image standard [8]. Decompression and grayscale conversion are accomplished via `libjpeg` [6].

As of version 8.0, `libjpeg` provided a way to handle JPEG images existing within in-memory buffers, as opposed to on-disk files. Because the `Image` class requires in-memory buffers, `JPEG` includes a JPEG memory source manager implementation, but it is built only if a version of `libjpeg` older than 8.0 is detected at compile-time.

`JPEG` provides a static function to determine whether or not a data buffer appears to be encoded in the JPEG image standard format. Errors within `libjpeg` will be caught and rethrown as `Exceptions`.

10.5 JPEGL

Similar to `JPEG`, the `JPEGL` class performs `Image` class services for lossless JPEG encoded images. `JPEGL` decompression is performed by NIST Biometric Image Software’s `libjpegl` [11].

10.6 JPEG2000

The `JPEG2000` class provides `Image` class functionality to JPEG 2000-encoded images [7]. The class makes an attempt to support the following JPEG 2000 codecs:

- JPEG 2000 codestream (.j2k)
- JPEG 2000 compressed image data (.jp2)

- JPEG 2000 interactive protocol (.jpt)

Decompression is provided by the OpenJPEG library (`libopenjpeg`) [10]. JPEG2000 also provides a static function to test whether or not an image appears to be JPEG 2000-encoded.

Not all information required by the `Image` class is present in a JPEG 2000-encoded image. In particular, some codecs and encoders omit the “Display Resolution Box.” It is generally accepted that the resolution will be 72 pixels-per-inch when the “Display Resolution Box” is not present.

Errors within `libopenjpeg` will be caught and rethrown as `Exceptions`.

10.7 NetPBM

The `NetPBM` class provides `Image` class functionality to all types of NetPBM formatted images, up to 48-bit depth. This includes the following formats:

- ASCII Portable Bitmap (P1, .pbm)
- ASCII Portable Graymap (P2, .pgm)
- ASCII Portable Pixmap (P3, .ppm)
- Binary Portable Bitmap (P4, .pbm)
- Binary Portable Graymap (P5, .pgm)
- Binary Portable Pixmap (P6, .ppm)

`NetPBM` provides some of its more general use parsing algorithms as static functions for use outside of the class. This includes ASCII to binary pixel conversion. A function to test for NetPBM formats is also provided.

10.8 PNG

The `PNG` class represents an image encoded according to the PNG image standard [5]. Decompression is provided by `libpng` [13].

`PNG` provides a static function to test whether or not an image appears to be encoded in the PNG image standard format. Errors within `libpng` are caught and rethrown as `Exceptions`.

10.9 WSQ

Images encoded in the WSQ-image standard [15] are represented by the `WSQ` class. The WSQ decompressor found in NIST Biometric Image Software [11], `libwsq`, is used by this class. The class provides a static function to determine whether or not an image appears to be encoded in the WSQ format.

Errors from the `libwsq` will be displayed through `stderr` and will **not** be rethrown as `Exceptions`.

Chapter 11

Text

The `Text` package consists of functions to perform common operations on `strings` and `char` arrays. Many of the operations may be considered “trivial,” but are used often enough within the Biometric Evaluation Framework and other applications that a common implementation in `BCommon` is more than warranted. A complete listing of functions is available in the documentation appendix for `BiometricEvaluation::Text`.

Listing 11.1 shows how to use the `split()` function from the `Text` package. `split()` can separate a string into tokens delimited by a character, useful for processing comma- or space-separated text files (such files could be produced by a `LogSheet` (Section 6.3 on page 17), for instance). Here, a text file containing metadata for an image is being parsed, perhaps to be passed to the `RawImage` constructor (Section 10.3 on page 32).

Listing 11.1: Tokenizing a string

```
1  /* Definition of input strings */
2  static const vector<string>::size_type filenameToken = 0;
3  static const vector<string>::size_type widthToken = 1;
4  static const vector<string>::size_type heightToken = 2;
5  static const vector<string>::size_type depthToken = 3;
6
7  /* Split the string, presumably input from a file */
8  string input = "/mnt/raw\\ images/1.raw 500 500 8";
9  vector<string> tokens = Text::split(input, ' ', true);
10
11 /* Assign the retrieved tokens */
12 string filename;
13 uint32_t width, height, depth;
14 try {
15     filename = tokens.at(filenameToken);    /* "/mnt/raw images/1.raw" */
16     width = atoi(tokens.at(widthToken).c_str());    /* "500" */
17     height = atoi(tokens.at(heightToken).c_str());    /* "500" */
18     depth = atoi(tokens.at(depthToken).c_str());    /* "8" */
19 } catch (out_of_range) {
20     throw Error::FileError("Malformed input");
21 }
```

Notice the `true` parameter to `split()` in Listing 11.1. This instructs `split()` to not tokenize based on an escaped delimiter. If `false`, the first token would be split into two at the presence of the delimiter.

Text also contains functions to perform hashing via OpenSSL. A two-line program that emulates the command-line md5sum program is shown in Listing 11.2. Changing the digest parameter to "sha1" would make the program emulate 'openssl sha1'.

Listing 11.2: md5sum via BECommon

```
1 #include <cstdlib>
2 #include <iostream>
3
4 #include <be_io_utility.h>
5 #include <be_text.h>
6 #include <be_memory_autoarray.h>
7
8 using namespace std;
9 using namespace BiometricEvaluation;
10
11 int
12 main(
13     int argc,
14     char *argv[])
15 {
16     if (argc == 0)
17         return (EXIT_FAILURE);
18
19     try {
20         Memory::uint8Array file = IO::Utility::readFile(argv[1]);
21         cout << Text::digest(file, file.size(), "md5") << " " <<
22             argv[1] << endl;
23     } catch (Error::Exception) {
24         return (EXIT_FAILURE);
25     }
26
27     return (EXIT_SUCCESS);
28 }
```

Chapter 12

Feature

The `Feature` package contains those items that relate to the representation of biometric features, such as fingerprint minutiae, facial features (eyes, etc.), and related information. Objects of these class types are typically associated with `View` (Chapter 14 on page 41) or `DataInterchange` (Chapter 15 on page 43) objects. For example, a minutiae object is usually obtained from a finger view, which may have been obtained from a data interchange object representing an entire biometric record for an individual.

The data contained within a `Feature` object is represented as the “native” format as it was extracted from the underlying data record. There is no translation to a common format and it is the application’s responsibility to interpret or translate the data as necessary.

12.1 ANSI/NIST Features

The ANSI/NIST [3] standard defines several features represented as data elements within a record. Fingerprint and palm minutiae is contained within Type-9 record. The `AN2K7Minutiae` class, contained in the `Feature` package, represents a single Type-9 record. An object of this class can be constructed directly from a complete ANSI/NIST record. However, it is more common for an application to retrieve these objects from the `AN2KView` object defined in the `Finger` package (Chapter 13 on page 39).

Chapter 13

Finger

One of the most commonly used biometric source is the fingerprint. Multiple types of information can be derived from a fingerprint, including minutiae and the pattern, such as whorl, etc. The `Finger` package contains the types, classes, and other items that are related to fingers and fingerprints. Objects of the `Finger` classes are typically not used in a stand-alone fashion, but are usually obtained from an object in the `DataInterchange` [15](#) package.

Several enumerated types are defined in the `Finger` package. The types are used to represent those elements related to fingers and fingerprints that are common across all data formats. Types that represent finger position, impression type, and others are included in the package. Stream operators are defined for these types so they can be printed in human-readable format.

Most of the classes in the `Finger` package represent data taken directly from a record in a standard format (e.g. ANSI/NIST [\[3\]](#)). In addition to general information, such as finger position, other information may be represented: The source of the finger image; the quality of the image, etc. In addition to this descriptive information, the finger object will provide the set of derived minutiae or other data sets.

When representing the information about a finger (and fingerprint), the class in the `Finger` package implements the interface defined in the `View` package. A finger is a specific type of view in that it represents all the available information about the finger, including the source image, minutiae (often in several formats), as well as the capture data (date, location, etc.) Finger views are documented in [Section 14.1 on page 41](#).

13.1 ANSI/NIST Minutiae Data Record

The `AN2KMinutiaeDataRecord` class represents all of the information taken from a ANSI/NIST Type-9 record. A Type-9 record may include minutiae data items in several formats (standard and proprietary) and the impression type code.

Chapter 14

View

Within the Biometric Evaluation Framework a `View` represents all the information that was derived from an image of a biometric sample. For example, with a fingerprint image, any minutiae that were extracted from that image, as well as the image itself, are contained within a single `View` object. In many cases the image may not be present, however the image size and other information is contained within a biometric record, along with the derived information. A `View` is used to represent these records as well.

`View` objects are created with information taken from a biometric data record, an ANSI/NIST 2007 file, for example. Most record formats contain information about the image itself, such as the resolution and size. The `View` object can be used to retrieve this information. However, the data may differ from that contained in the image itself, and applications can compare the corresponding values taken from the `Image` object (when available) to those taken from the `View` object.

In the case where a raw image is part of the biometric record, the `View` object's related `Image` object will have identical size, resolution, etc. values because the `View` class sets the `Image` attributes directly. For other image types (e.g. JPEG) the `Image` object will return attribute values taken from the image data.

14.1 Finger Views

Finger views are objects that represent all the available information for a specific finger as contained in one or more biometric records. For example, an ANSI/NIST file may contain a Type-3 record (finger image) and an associated Type-9 record (finger minutiae). A finger view object based on these two records can be instantiated and used by an application to retrieve all the desired information, including the source finger image. The internals of record processing and error handling are encapsulated within the class.

The `BECommon` provides several classes that are derived from a base `View` class, contained within the `Finger` package. See [Chapter 13 on page 39](#) for more information on the types associated with fingers and fingerprints. This section discusses finger views, the classes which are derived from the general `View` class. These subclasses represent specific biometric file types, such as ANSI/NIST or INCITS/M1. In the latter case, two files must be provided when constructing the object because INCITS finger image and finger minutiae records are defined in two separate standards.

14.1.1 ANSI/NIST Finger Views

Listing 14.1: Using an AN2K Finger View

```

1 #include <fstream>
2 #include <iostream>
3 #include <be_finger_an2kview_fixedres.h>
4 using namespace std;
5 using namespace BiometricEvaluation;
6
7 int
8 main(int argc, char* argv[]) {
9
10     Finger::AN2KViewFixedResolution *_an2kv
11     try {
12         _an2kv = new Finger::AN2KViewFixedResolution("type9-3.an2k",
13             TYPE_3_ID, 1);
14     } catch (Error::DataError &e) {
15         cerr << "Caught " << e.getInfo() << endl;
16         return (EXIT_FAILURE);
17     } catch (Error::FileError& e) {
18         cerr << "A file error occurred: " << e.getInfo() << endl;
19         return (EXIT_FAILURE);
20     }
21     std::auto_ptr<Finger::AN2KView> an2kv(_an2kv);
22
23     cout << "Image resolution is " << an2kv->getImageResolution() << endl;
24     cout << "Image size is " << an2kv->getImageSize() << endl;
25     cout << "Image depth is " << an2kv->getImageDepth() << endl;
26     cout << "Compression is " << an2kv->getCompressionAlgorithm() << endl;
27     cout << "Scan resolution is " << an2kv->getScanResolution() << endl;
28
29     // Save the finger image to a file.
30     trl::shared_ptr<Image::Image> img = an2kv->getImage();
31     if (img.get() == NULL) {
32         cerr << "Image was not present." << endl;
33         return (EXIT_FAILURE);
34     }
35     string filename = "rawimg";
36     ofstream img_out(filename.c_str(), ofstream::binary);
37     img_out.write((char *)&(img->getRawData()[0]),
38         img->getRawData().size());
39     if (img_out.good())
40         cout << "\tFile: " << filename << endl;
41     else {
42         img_out.close();
43         cerr << "Error occurred when writing " << filename << endl;
44         return (EXIT_FAILURE);
45     }
46     img_out.close();
47
48     // Get the finger minutiae sets. AN2K records can have more than one
49     // set of minutiae for a finger.
50
51     vector<Finger::AN2KMinutiaeDataRecord> mindata = an2kv->getMinutiaeDataRecordSet();
52 }

```

14.1.2 ISO/INCITS Finger Views

Chapter 15

Data Interchange

The `DataInterchange` package consists of classes and other elements used to process an entire biometric data record. For example, a single ANSI/NIST record, consisting of many smaller records (fingerprint images, latent data, etc.) can be accessed by instantiating a single object. Classes in this package typically use has-a relationships to classes in the `Finger` and other packages that process individual biometric samples.

The design of classes in the `DataInterchange` package allows applications to create a single object from a biometric record, such as an ANSI/NIST file. After creating this object, the application can retrieve the needed information (such as fingerprint images) from this object, or objects returned by methods of the `DataInterchange` object. A typical example would be to retrieve all images from the record and pass them into a function that extracts a biometric template or some other image processing.

15.1 ANSI/NIST Data Records

The ANSI/NIST Data Interchange package contains the classes used to represent ANSI/NIST [3] records. One class, `AN2KRecord`, is used to represent the entire ANSI/NIST record. An object of this class will contain objects of the `Finger` classes, as well as other packages. By instantiating the `AN2KRecord` object, the application can retrieve all the information and images contained in the ANSI/NIST record.

The `AN2KMinutiaeDataRecord` class represents an entire Type-9 record from an ANSI/NIST file. However, some components of this class are represented by classes in other packages. For example, the `AN2K7Minutiae` class in the `Feature` package represents the “standard” format minutiae in the Type-9 record.

Listing 15.1 shows how an application can retrieve all finger captures (Type-4 records) from an ANSI/NIST record. Once the Views are retrieved, the application obtains the set of minutiae records associated with that View.

Listing 15.1: Retrieving ANSI/NIST Records

```
1 #include <iostream>
2 #include <be_error_exception.h>
3 #include <be_finger_an2kview_capture.h>
4
5 int
6 main(int argc, char* argv[])
7 {
8     /*
9      * Call the constructor that will open an existing AN2K file and
```

```

10      * retrieve the first finger capture (Type-14) record.
11      */
12      std::auto_ptr<Finger::AN2KViewCapture> an2kv;
13      try {
14          an2kv.reset(new Finger::AN2KViewCapture("type9-14.an2k", 1));
15      } catch (Error::DataError &e) {
16          cout << "Caught " << e.getInfo() << endl;
17          return (EXIT_FAILURE);
18      } catch (Error::FileError& e) {
19          cout << "A file error occurred: " << e.getInfo() << endl;
20          return (EXIT_FAILURE);
21      }
22
23      cout << "Get the set of minutiae data records: ";
24      vector<Finger::AN2KMinutiaeDataRecord> records =
25          an2kv->getMinutiaeDataRecordSet();
26      cout << "There are " << records.size() << " minutiae records." << endl;
27
28      /*
29       * Get the info from the first minutiae record in the View.
30       */
31      DataInterchange::AN2KMinutiaeDataRecord type9 = records[0];
32
33      /*
34       * Get the "standard" set of minutiae.
35       */
36      Feature::AN2K7Minutiae an2k7m = type9.getAN2K7Minutiae();
37
38      /*
39       * Obtain the minutiae points, ridge counts, cores, and deltas.
40       */
41      Feature::MinutiaPointSet mps;
42      Feature::RidgeCountItemSet rcs;
43      Feature::CorePointSet cps;
44      Feature::DeltaPointSet dps;
45      try {
46          mps = an2k7m->getMinutiaPoints();
47          rcs = an2k7m->getRidgeCountItems();
48          cps = an2k7m->getCores();
49          dps = an2k7m->getDeltas();
50
51      } catch (Error::DataError &e) {
52          cout << "Caught " << e.getInfo() << endl;
53          return (EXIT_FAILURE);
54      }
55
56      cout << "There are " << mps.size() << " minutiae points:" << endl;
57      /*
58       * Print out the minutiae points.
59       */
60      for (int i = 0; i < mps.size(); i++) {
61          printf("(%u,%u,%u)\n", mps[i].coordinate.x, mps[i].coordinate.y,
62              mps[i].theta);
63      }
64      cout << "There are " << rcs.size() << " ridge counts:" << endl;
65      for (int i = 0; i < rcs.size(); i++) {

```

```

66         printf("(%u,%u,%u)\n", rcs[i].index_one, rcs[i].index_two,
67         rcs[i].count);
68     }
69     cout << "There are " << cps.size() << " cores." << endl;
70     cout << "There are " << dps.size() << " deltas." << endl;
71
72     cout << "Fingerprint Reader: " << endl;
73     try { cout << an2k7m->getOriginatingFingerprintReadingSystem() << endl; }
74     catch (Error::ObjectDoesNotExist) { cout << "<Omitted>" << endl; }
75
76     cout << "Pattern (primary): " <<
77     Feature::AN2K7Minutiae::convertPatternClassification(
78     an2k7m->getPatternClassificationSet().at(0)) << endl;
79
80     return (EXIT_SUCCESS);
81 }

```

Listing 15.2 shows how an application can retrieve all latent finger images from a set of ANSI/NIST record retrieved from a RecordStore. Using the Image object, the image's "raw" data can be retrieved and passed to another function for processing. Note that the image data may be stored in a compressed format inside the ANSI/NIST record, but is converted to raw format by the Image object.

Listing 15.2: Retrieving ANSI/NIST Records

```

1 #include <be_io_recordstore.h>
2 #include <be_data_interchange_an2k.h>
3 using namespace BiometricEvaluation;
4
5 void
6 processImageData(uint8_t *buf, uint32_t size)
7 {
8     :
9     :
10    :
11    :
12 }
13
14 int
15 main(int argc, char* argv[]) {
16
17     std::tr1::shared_ptr<IO::RecordStore> rs;
18     try {
19         rs = IO::RecordStore::openRecordStore(rsname, datadir, IO::READONLY);
20     } catch (Error::Exception &e) {
21         cerr << "Could not open record store: " << e.getInfo() << endl;
22         return (EXIT_FAILURE);
23     }
24
25     /*
26      * Read some AN2K records and construct the View objects.
27      */
28     Utility::uint8Array data;
29     string key;
30     while (true) { // Loop through all records in store
31         uint64_t rlen;
32         try {

```

```

33         rlen = rs->sequence(key, NULL);
34     } catch (Error::ObjectDoesNotExist &e) {
35         break;
36     } catch (Error::Exception &e) {
37         cout << "Failed sequence: " << e.getInfo() << endl;
38         return (EXIT_FAILURE);
39     }
40     data.resize(rlen);
41     try {
42         rs->read(key, data);
43         DataInterchange::AN2KRecord an2k(data);
44         std::vector<Finger::AN2KViewLatent> latents = an2k.getFingerLatents();
45         for (int i = 0; i < latents.size(); i++) {
46             tr1::shared_ptr<Image::Image> img = latents[i].getImage();
47             if (img != NULL) {
48                 cout << "\tCompression: " << img->getCompressionAlgorithm() << endl;
49                 cout << "\tDimensions: " << img->getDimensions() << endl;
50                 cout << "\tResolution: " << img->getResolution() << endl;
51                 cout << "\tDepth: " << img->getDepth() << endl;
52                 processImageData(img->getRawData(), img->getRawData().size());
53             }
54         }
55     } catch (Error::Exception &e) {
56         return (EXIT_FAILURE);
57     }
58 }
59 return (EXIT_SUCCESS);
60 }

```

15.2 INCITS Data Records

This INCITS class of data records covers all those record formats that are derived from the standards defined by the InterNational Committee for Information Technology Standards [9]. These formats include the ANSI-2004 Finger Minutiae Record Format [1], the ISO equivalent [2], and other data formats, including finger images.

15.2.1 Finger Views

Within the BECommon, finger view objects (Section 14.1) can be created from a combination of finger minutiae and image records. However, it is not necessary to have both records in order to create the view because each record contains enough information to represent the finger (image size, for example). However, if a view is constructed using only the minutiae record, then the image itself will not be present. Alternatively, if a view is made from an image record, no minutiae data would be available. It is possible to construct a view without any information.

Listing 15.3 shows an example of accessing the information in an ANSI 378-2004 Finger Minutiae Record by creating an ANSI2004View object from the record file.

Listing 15.3: INCITS Finger Views

```

1 #include <be_finger_ansi2004view.h>
2 using namespace BiometricEvaluation;
3

```

```

4 int
5 main(int argc, char* argv[])
6 {
7     /*
8      * Create a finger view from a file containing an ANSI-2004 finger
9      * minutiae record, and without the finger view record. Read the
10     * third finger view from the file.
11     */
12     Finger::ANSI2004View fngv;
13     try {
14         fngv = Finger::ANSI2004View("fmr.ansi2004", "", 3);
15     } catch (Error::DataError &e) {
16         cout << "Caught " << e.getInfo() << endl;
17         exit (EXIT_FAILURE);
18     } catch (Error::FileError& e) {
19         cout << "A file error occurred: " << e.getInfo() << endl;
20         exit (EXIT_FAILURE);
21     }
22
23     /*
24      * The ANSI2004View implementation of the View::View interface.
25      */
26     cout << "Image resolution is " << fngv.getImageResolution() << endl;
27     cout << "Image size is " << fngv.getImageSize() << endl;
28     cout << "Image depth is " << fngv.getImageDepth() << endl;
29     cout << "Compression is " << fngv.getCompressionAlgorithm() << endl;
30     cout << "Scan resolution is " << fngv.getScanResolution() << endl;
31
32     /*
33      * Test the ANSI2004View implementation of the Finger::INCITSVIEW
34      * interface.
35      */
36     cout << "Finger position is " << fngv.getPosition() << endl;
37     cout << "Impression type is " << fngv.getImpressionType() << endl;
38     cout << "Quality is " << fngv.getQuality() << endl;
39     cout << "Eqpt ID is " << hex << showbase << fngv.getCaptureEquipmentID() << endl;
40     cout << dec;
41
42     Feature::INCITSMinutiae fmd = fngv.getMinutiaeData();
43     cout << "Minutiae format is " << fmd.getFormat() << endl;
44     Feature::MinutiaPointSet mps = fmd.getMinutiaPoints();
45     cout << "There are " << mps.size() << " minutiae points:" << endl;
46     for (int i = 0; i < mps.size(); i++)
47         cout << mps[i];
48
49     Feature::RidgeCountItemSet rcs = fmd.getRidgeCountItems();
50     cout << "There are " << rcs.size() << " ridge count items:" << endl;
51     for (int i = 0; i < rcs.size(); i++)
52         cout << "\t" << rcs[i];
53
54     Feature::CorePointSet cores = fmd.getCores();
55     cout << "There are " << cores.size() << " cores:" << endl;
56     for (int i = 0; i < cores.size(); i++)
57         cout << "\t" << cores[i];
58
59     Feature::DeltaPointSet deltas = fmd.getDeltas();

```

```
60 |     cout << "There are " << deltas.size() << " deltas:" << endl;
61 |     for (int i = 0; i < deltas.size(); i++)
62 |         cout << "\t" << deltas[i];
63 |
64 |     exit (EXIT_SUCCESS);
65 | }
```

Bibliography

- [1] *ANSI INCITS 378-2004: Finger Minutiae Format for Data Interchange*. ANSI/INCITS, 2004. 46
- [2] *ISO/IEC 19794-2: Information technology - Biometric data interchange formats - Part 2: Finger minutiae data*. ISO/IEC, first edition, 2005. 46
- [3] *American National Standard for Information Systems - Data Format for the Interchange of Fingerprint, Facial, & Other Biometric Information*. ANSI/NIST-ITL, 1-2007 edition, 2007. 37, 39, 43
- [4] Mark Adler. zlib, 2012. <http://www.zlib.net/>. 19
- [5] World Wide Web Consortium. Portable Network Graphics Standard, 2003. <http://www.w3.org/TR/PNG/>. 33
- [6] Independent JPEG Group. libjpeg, 2011. <http://www.ijg.org/>. 32
- [7] Joint Photographic Experts Group. JPEG2000 Image Standard, 1992. <http://www.jpeg.org/jpeg2000/index.html>. 32
- [8] Joint Photographic Experts Group. JPEG Image Standard, 2011. <http://www.jpeg.org/jpeg/index.html>. 32
- [9] InterNational Committee for Information Technology Standards. <http://www.incits.org>. 46
- [10] Communications and Remote Sensing Lab, Université catholique de Louvain. OpenJPEG Library, 2011. <http://www.openjpeg.org/>. 33
- [11] NIST Biometric Image Software, 2011. <http://www.nist.gov/itl/iad/ig/nbis.cfm>. 7, 32, 33
- [12] NIST Image Group. <http://www.nist.gov/itl/iad/ig/>. 1
- [13] Greg Roelofs. libpng, 2011. <http://www.libpng.org/pub/png/libpng.html>. 33
- [14] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, special edition, 2000. 3, 9
- [15] Wavelet Scalar Quantization Gray-Scale Fingerprint Image Compression Standard, 2010. https://www.fbibiospecs.org/docs/WSQ_Gray-scale_Specification_Version_3_1_Final.pdf. 33

Appendix A

Namespace Index

A.1 Namespace List

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

BiometricEvaluation::Error	
Exceptions, and other error handling	61
BiometricEvaluation::Finger	
Biometric information relating to finger images and derived information	62
BiometricEvaluation::Framework	
Information about the framework	64
BiometricEvaluation::Image	
Basic information relating to images	65
BiometricEvaluation::IO	
Input/Output functionality	67
BiometricEvaluation::IO::Utility	69
BiometricEvaluation::Memory	
Support for memory-related operations	76
BiometricEvaluation::Process	
Process information and controls	76
BiometricEvaluation::System	
Operating system, hardware, etc	77
BiometricEvaluation::Text	
Text processing for string objects	79
BiometricEvaluation::Time	
Support for time and timers	81
BiometricEvaluation::View	
View information	82

Appendix B

Class Index

B.1 Class Hierarchy

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

BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged	83
BiometricEvaluation::Finger::AN2KMinutiaeDataRecord	88
BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric	90
BiometricEvaluation::DataInterchange::AN2KRecord	91
BiometricEvaluation::Memory::AutoArray< T >	132
BiometricEvaluation::Memory::AutoBuffer< T >	139
BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet	140
BiometricEvaluation::Image::CompressionAlgorithm	148
BiometricEvaluation::IO::Compressor	148
BiometricEvaluation::IO::GZip	191
BiometricEvaluation::Image::Coordinate	160
BiometricEvaluation::Feature::CorePoint	161
BiometricEvaluation::Feature::DeltaPoint	168
BiometricEvaluation::View::AN2KView::DeviceMonitoringMode	169
BiometricEvaluation::DataInterchange::AN2KRecord::DomainName	170
BiometricEvaluation::Feature::AN2K7Minutiae::EncodingMethod	171
BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry	171
BiometricEvaluation::Error::Exception	172
BiometricEvaluation::Error::ConversionError	159
BiometricEvaluation::Error::DataError	161
BiometricEvaluation::Error::FileError	174
BiometricEvaluation::Error::MemoryError	249
BiometricEvaluation::Error::NotImplemented	257
BiometricEvaluation::Error::ObjectDoesNotExist	258
BiometricEvaluation::Error::ObjectExists	259
BiometricEvaluation::Error::ObjectIsClosed	260
BiometricEvaluation::Error::ObjectIsOpen	261
BiometricEvaluation::Error::ParameterError	262
BiometricEvaluation::Error::StrategyError	318
BiometricEvaluation::Finger::FingerImageCode	181
BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem	182

BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition	182
BiometricEvaluation::Image::Image	198
BiometricEvaluation::Image::JPEG	228
BiometricEvaluation::Image::JPEG2000	230
BiometricEvaluation::Image::JPEGL	233
BiometricEvaluation::Image::NetPBM	252
BiometricEvaluation::Image::PNG	264
BiometricEvaluation::Image::Raw	281
BiometricEvaluation::Image::WSQ	332
BiometricEvaluation::Finger::Impression	206
BiometricEvaluation::Memory::IndexedBuffer	221
BiometricEvaluation::IO::LogCabinet	235
BiometricEvaluation::IO::LogSheet	237
BiometricEvaluation::Process::Manager	245
BiometricEvaluation::Process::ForkManager	183
BiometricEvaluation::Process::POSIXThreadManager	267
BiometricEvaluation::IO::ManifestEntry	249
BiometricEvaluation::Feature::Minutiae	250
BiometricEvaluation::Feature::AN2K7Minutiae	84
BiometricEvaluation::Feature::INCITSMinutiae	207
BiometricEvaluation::Feature::MinutiaeFormat	251
BiometricEvaluation::Feature::MinutiaeType	252
BiometricEvaluation::Feature::MinutiaPoint	252
BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification	263
BiometricEvaluation::Finger::PatternClassification	264
BiometricEvaluation::Finger::Position	266
BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate	272
BiometricEvaluation::IO::Properties	273
BiometricEvaluation::IO::PropertiesFile	279
BiometricEvaluation::IO::RecordStore	283
BiometricEvaluation::IO::ArchiveRecordStore	124
BiometricEvaluation::IO::CompressedRecordStore	141
BiometricEvaluation::IO::DBRecordStore	162
BiometricEvaluation::IO::FileRecordStore	175
BiometricEvaluation::IO::SQLiteRecordStore	306
BiometricEvaluation::View::AN2KView::RecordType	298
BiometricEvaluation::Image::Resolution	299
BiometricEvaluation::Feature::RidgeCountExtractionMethod	300
BiometricEvaluation::Feature::RidgeCountItem	301
BiometricEvaluation::Error::SignalManager	301
BiometricEvaluation::Image::Size	305
BiometricEvaluation::Process::Statistics	314
BiometricEvaluation::Time::Timer	319
BiometricEvaluation::View::View	320
BiometricEvaluation::Finger::INCITSView	210
BiometricEvaluation::Finger::ANSI2004View	118
BiometricEvaluation::Finger::ANSI2007View	120
BiometricEvaluation::Finger::ISO2005View	226
BiometricEvaluation::View::AN2KView	100
BiometricEvaluation::Finger::AN2KView	97

BiometricEvaluation::Finger::AN2KViewFixedResolution	109
BiometricEvaluation::View::AN2KViewVariableResolution	115
BiometricEvaluation::Finger::AN2KViewVariableResolution	112
BiometricEvaluation::Finger::AN2KViewCapture	105
BiometricEvaluation::Finger::AN2KViewLatent	111
BiometricEvaluation::Time::Watchdog	322
BiometricEvaluation::Process::Worker	325
BiometricEvaluation::Process::WorkerController	329
BiometricEvaluation::Process::ForkWorkerController	187
BiometricEvaluation::Process::POSIXThreadWorkerController	270

Appendix C

Class Index

C.1 Class List

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

BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged	
Amputated or bandaged code	83
BiometricEvaluation::Feature::AN2K7Minutiae	
A class to represent a set of minutiae in an ANSI/NIST record	84
BiometricEvaluation::Finger::AN2KMinutiaeDataRecord	
Representation of a Type-9 Record from an AN2K file	88
BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric	
A structure to represent an AN2K quality metric	90
BiometricEvaluation::DataInterchange::AN2KRecord	
A class to represent an entire ANSI/NIST record	91
BiometricEvaluation::Finger::AN2KView	
A class to represent single finger view and derived information	97
BiometricEvaluation::View::AN2KView	
A class to represent single biometric view and derived information	100
BiometricEvaluation::Finger::AN2KViewCapture	
Represents an ANSI/NIST variable-resolution finger image	105
BiometricEvaluation::Finger::AN2KViewFixedResolution	
A class to represent single finger view and derived information	109
BiometricEvaluation::Finger::AN2KViewLatent	
.	111
BiometricEvaluation::Finger::AN2KViewVariableResolution	
A class to represent single finger view based on an ANSI/NIST record	112
BiometricEvaluation::View::AN2KViewVariableResolution	
A class to represent single view based on an ANSI/NIST record	115
BiometricEvaluation::Finger::ANSI2004View	
A class to represent single finger view and derived information	118
BiometricEvaluation::Finger::ANSI2007View	
A class to represent single finger view and derived information	120
BiometricEvaluation::IO::ArchiveRecordStore	
This class implements the IO::RecordStore interface by storing data items in single file, with an associated manifest file	124

BiometricEvaluation::Memory::AutoArray< T >	
A C-style array wrapped in the facade of a C++ STL container	132
BiometricEvaluation::Memory::AutoBuffer< T >	139
BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet	140
BiometricEvaluation::IO::CompressedRecordStore	
Sibling-implemented RecordStore with Compression	141
BiometricEvaluation::Image::CompressionAlgorithm	
Image compression algorithms	148
BiometricEvaluation::IO::Compressor	148
BiometricEvaluation::Error::ConversionError	
Error when converting one object into another, a property value from string to int, for example	159
BiometricEvaluation::Image::Coordinate	
A structure to contain a two-dimensional coordinate without a specified origin	160
BiometricEvaluation::Feature::CorePoint	
Representation of the core	161
BiometricEvaluation::Error::DataError	
Error when reading data from an external source	161
BiometricEvaluation::IO::DBRecordStore	
A class that implements IO::RecordStore using a Berkeley DB database as the underlying record storage system	162
BiometricEvaluation::Feature::DeltaPoint	
Representation of the delta	168
BiometricEvaluation::View::AN2KView::DeviceMonitoringMode	
The level of human monitoring for the image capture device	169
BiometricEvaluation::DataInterchange::AN2KRecord::DomainName	
Representation of a domain name for the user-defined Type-2 logical record implementation	170
BiometricEvaluation::Feature::AN2K7Minutiae::EncodingMethod	
Methods for encoding minutiae data in an AN2K record	171
BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry	171
BiometricEvaluation::Error::Exception	
The parent class of all BiometricEvaluation exceptions	172
BiometricEvaluation::Error::FileError	
File error when opening, reading, writing, etc	174
BiometricEvaluation::IO::FileRecordStore	175
BiometricEvaluation::Finger::FingerImageCode	181
BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem	
Representation of information about a fingerprint reader system	182
BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition	
Locations of an individual finger segment in a slap	182
BiometricEvaluation::Process::ForkManager	
Manager implementation that starts Workers by calling <code>fork(2)</code>	183
BiometricEvaluation::Process::ForkWorkerController	
Wrapper of a Worker returned from a Process::ForkManager	187
BiometricEvaluation::IO::GZip	
Compressor for gzip compression from <code>zlib</code>	191
BiometricEvaluation::Image::Image	
Represent attributes common to all images	198
BiometricEvaluation::Finger::Impression	
Finger and palm impression types	206

BiometricEvaluation::Feature::INCITSMinutiae	
A class to represent a set of minutiae in an ANSI/INCITS record	207
BiometricEvaluation::Finger::INCITSView	
A class to represent single finger view and derived information	210
BiometricEvaluation::Memory::IndexedBuffer	
Manage a memory buffer with an index	221
BiometricEvaluation::Finger::ISO2005View	
A class to represent single finger view and derived information	226
BiometricEvaluation::Image::JPEG	
A JPEG-encoded image	228
BiometricEvaluation::Image::JPEG2000	
A JPEG-2000-encoded image	230
BiometricEvaluation::Image::JPEGL	
A Lossless JPEG-encoded image	233
BiometricEvaluation::IO::LogCabinet	235
BiometricEvaluation::IO::LogSheet	
A class to represent a single logging mechanism	237
BiometricEvaluation::Process::Manager	
An interface for intranode process management classes	245
BiometricEvaluation::IO::ManifestEntry	249
BiometricEvaluation::Error::MemoryError	
An error occurred when allocating an object	249
BiometricEvaluation::Feature::Minutiae	
A class to represent a set of minutiae data points	250
BiometricEvaluation::Feature::MinutiaeFormat	
Enumerate the minutiae format standards	251
BiometricEvaluation::Feature::MinutiaeType	
Enumerate the types of minutiae: Ridge Ending, Bifurcation, Compound, or other	252
BiometricEvaluation::Feature::MinutiaPoint	
Representation of a finger minutiae data point	252
BiometricEvaluation::Image::NetPBM	
A NetPBM-encoded image	252
BiometricEvaluation::Error::NotImplemented	
A NotImplemented object is thrown when the underlying implementation of this interface has not or could not be created	257
BiometricEvaluation::Error::ObjectDoesNotExist	
The named object does not exist	258
BiometricEvaluation::Error::ObjectExists	
The named object exists and will not be replaced	259
BiometricEvaluation::Error::ObjectIsClosed	
The object is closed	260
BiometricEvaluation::Error::ObjectIsOpen	
The object is already opened	261
BiometricEvaluation::Error::ParameterError	
An invalid parameter was passed to a constructor or method	262
BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification	
Pattern classification codes	263
BiometricEvaluation::Finger::PatternClassification	
Pattern classification codes	264
BiometricEvaluation::Image::PNG	
A PNG-encoded image	264

BiometricEvaluation::Finger::Position	
Finger position codes	266
BiometricEvaluation::Process::POSIXThreadManager	
Manager implementation that starts Workers in POSIX threads	267
BiometricEvaluation::Process::POSIXThreadWorkerController	
Decorated Worker returned from a Process::POSIXThreadManager	270
BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate	
Offsets to the bounding boxes for the EJI, full finger views, or EJI segments	272
BiometricEvaluation::IO::Properties	
Maintain key/value pairs of strings, with each property matched to one value	273
BiometricEvaluation::IO::PropertiesFile	
A Properties object persisted in an file on disk	279
BiometricEvaluation::Image::Raw	
An image with no encoding or compression	281
BiometricEvaluation::IO::RecordStore	
A class to represent a data storage mechanism	283
BiometricEvaluation::View::AN2KView::RecordType	
The type of AN2K record	298
BiometricEvaluation::Image::Resolution	
A structure to represent the resolution of an image	299
BiometricEvaluation::Feature::RidgeCountExtractionMethod	
Enumerate the types of extraction methods for ridge counts	300
BiometricEvaluation::Feature::RidgeCountItem	
Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number	301
BiometricEvaluation::Error::SignalManager	
A SignalManager object is used to handle signals that come from the operating system	301
BiometricEvaluation::Image::Size	
A structure to represent the size of an image, in pixels	305
BiometricEvaluation::IO::SQLiteRecordStore	
A RecordStore implementation using a SQLite database as the underlying record storage system	306
BiometricEvaluation::Process::Statistics	
Interface for gathering process statistics, such as memory usage, system time, etc	314
BiometricEvaluation::Error::StrategyError	
A StrategyError object is thrown when the underlying implementation of this interface encounters an error	318
BiometricEvaluation::Time::Timer	
This class can be used by applications to report the amount of time a block of code takes to execute	319
BiometricEvaluation::View::View	
A class to represent single biometric element view	320
BiometricEvaluation::Time::Watchdog	
A Watchdog object can be used by applications to limit the amount of processing time taken by a block of code	322
BiometricEvaluation::Process::Worker	
An abstraction of an instance that performs work on given data	325
BiometricEvaluation::Process::WorkerController	
Wrapper of a Worker returned from a Process::Manager	329
BiometricEvaluation::Image::WSQ	
A WSQ-encoded image	332

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.

- class [SignalManager](#)

A [SignalManager](#) object is used to handle signals that come from the operating system.

Functions

- string [errorStr](#) ()
- void [SignalManagerSigHandler](#) (int signo, siginfo_t *info, void *uap)

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](#)

- class [AN2KMinutiaeDataRecord](#)

Representation of a Type-9 Record from an AN2K file.

- class [AN2KView](#)

A class to represent single finger view and derived information.

- class [AN2KViewCapture](#)

Represents an ANSI/NIST variable-resolution finger image.

- class [AN2KViewFixedResolution](#)

A class to represent single finger view and derived information.

- class [AN2KViewLatent](#)
- class [AN2KViewVariableResolution](#)
A class to represent single finger view based on an ANSI/NIST record.
- class [ANSI2004View](#)
A class to represent single finger view and derived information.
- class [ANSI2007View](#)
A class to represent single finger view and derived information.
- class [INCITSView](#)
A class to represent single finger view and derived information.
- class [ISO2005View](#)
A class to represent single finger view and derived information.

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 &)
- std::ostream & [operator<<](#) (std::ostream &stream, const [AN2KViewCapture::AmputatedBandaged::Kind](#) &ab)
Output stream overload for AmputatedBandaged::Kind.
- std::ostream & [operator<<](#) (std::ostream &stream, const [AN2KViewCapture::FingerSegmentPosition](#) &fsp)
Output stream overload for FingerSegmentPosition.
- std::ostream & [operator<<](#) (std::ostream &stream, const [AN2KViewVariableResolution::PrintPositionCoordinate](#) &ppc)
Output stream overload for PrintPositionCoordinate.

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.2.2 Function Documentation

D.2.2.1 `std::ostream& BiometricEvaluation::Finger::operator<< (std::ostream & stream, const AN2KViewVariableResolution::PrintPositionCoordinate & ppc)`

Output stream overload for PrintPositionCoordinate.

Parameters

<code>in</code>	<code><i>stream</i></code>	Stream on which to append formatted PrintPositionCoordinate information.
<code>in</code>	<code><i>ppc</i></code>	PrintPositionCoordinate information to append to stream.

Returns

stream with a ppc textual representation appended.

D.3 BiometricEvaluation::Framework Namespace Reference

Information about the framework.

Functions

- unsigned int [getMajorVersion \(\)](#)
Framework major version.
- unsigned int [getMinorVersion \(\)](#)
Framework minor version.
- std::string [getCompiler \(\)](#)
Compiler used to compile this framework.
- std::string [getCompileDate \(\)](#)
Date when this framework was compiled.
- std::string [getCompileTime \(\)](#)
Time when this framework was compiled.
- std::string [getCompilerVersion \(\)](#)
Version string of compiler used to compile this framework.

D.3.1 Detailed Description

Information about the framework.

D.3.2 Function Documentation

D.3.2.1 `unsigned int BiometricEvaluation::Framework::getMajorVersion ()`

[Framework](#) major version.

Returns

The major version number of the BiometricFramework

D.3.2.2 unsigned int BiometricEvaluation::Framework::getMinorVersion ()

Framework minor version.

Returns

The minor version of the BiometricEvaluation framework.

D.3.2.3 std::string BiometricEvaluation::Framework::getCompiler ()

Compiler used to compile this framework.

Returns

The name of the compiler used to compile this framework.

D.3.2.4 std::string BiometricEvaluation::Framework::getCompileDate ()

Date when this framework was compiled.

Returns

Date when this framework was compiled, in the form "MMM DD YYYY"

D.3.2.5 std::string BiometricEvaluation::Framework::getCompileTime ()

Time when this framework was compiled.

Returns

Time when this framework was compiled, in the form "HH:MM:SS"

D.3.2.6 std::string BiometricEvaluation::Framework::getCompilerVersion ()

Version string of compiler used to compile this framework.

Returns

Major, minor, and patch level of the compiler used.

D.4 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 [JPEG](#)
A JPEG-encoded image.
- class [JPEG2000](#)
A JPEG-2000-encoded image.
- class [JPEGL](#)
A Lossless JPEG-encoded image.
- class [NetPBM](#)
A NetPBM-encoded image.
- class [PNG](#)
A PNG-encoded image.
- class [Raw](#)
An image with no encoding or compression.
- class [WSQ](#)
A WSQ-encoded image.

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.4.1 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.4.2 Function Documentation

D.4.2.1 `std::ostream& BiometricEvaluation::Image::operator<< (std::ostream & stream, const CoordinateSet & coordinates)`

Output stream overload for CoordinateSet.

Parameters

<code>in</code>	<code><i>stream</i></code>	Stream on which to append formatted CoordinateSet information.
<code>in</code>	<code><i>coordinates</i></code>	CoordinateSet information to append to stream.

Returns

stream with a coordinates textual representation appended.

D.4.2.2 `float BiometricEvaluation::Image::distance (const Coordinate & p1, const Coordinate & p2)`

Calculate the distance between two points.

Parameters

<code>in</code>	<code><i>p1</i></code>	First point.
<code>in</code>	<code><i>p2</i></code>	Second point.

Returns

Distance between p1 and p2.

D.5 BiometricEvaluation::IO Namespace Reference

Input/Output functionality.

Namespaces

- namespace [Utility](#)

Classes

- struct [ManifestEntry](#)

- class [ArchiveRecordStore](#)
This class implements the [IO::RecordStore](#) interface by storing data items in single file, with an associated manifest file.
- class [CompressedRecordStore](#)
Sibling-implemented [RecordStore](#) with Compression.
- class [Compressor](#)
- class [DBRecordStore](#)
A class that implements [IO::RecordStore](#) using a Berkeley DB database as the underlying record storage system.
- class [FileRecordStore](#)
- class [GZip](#)
Compressor for gzip compression from zlib.
- class [LogCabinet](#)
- class [LogSheet](#)
A class to represent a single logging mechanism.
- class [Properties](#)
Maintain key/value pairs of strings, with each property matched to one value.
- class [PropertiesFile](#)
A [Properties](#) object persisted in an file on disk.
- class [RecordStore](#)
A class to represent a data storage mechanism.
- class [SQLiteRecordStore](#)
A [RecordStore](#) implementation using a SQLite database as the underlying record storage system.

Typedefs

- typedef map< string, [ManifestEntry](#) > [ManifestMap](#)
- typedef map< string, string > [PropertiesMap](#)

D.5.1 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.5.2 Typedef Documentation

D.5.2.1 typedef map<string, ManifestEntry> BiometricEvaluation::IO::ManifestMap

Convenience typedef for storing the manifest

D.5.2.2 typedef map<string, string> BiometricEvaluation::IO::PropertiesMap

Internal structure used for storing property keys/values

D.6 BiometricEvaluation::IO::Utility Namespace Reference

Functions

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

D.6.1 Detailed Description

A class containing utility functions used for [IO](#) operations. These functions are class methods.

D.6.2 Function Documentation

D.6.2.1 void BiometricEvaluation::IO::Utility::removeDirectory (const string & *directory*, const string & *prefix*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Remove a directory using directory name and parent pathname.

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

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

D.6.2.2 void BiometricEvaluation::IO::Utility::removeDirectory (const string & *pathname*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Remove a directory using a complete pathname.

Parameters

in	<i>pathname</i>	The complete path name of the directory to be removed,
----	-----------------	--

Exceptions

<i>Error::ObjectDoesNotExist</i>	The named directory does not exist.
<i>Error::StrategyError</i>	An error occurred when using the underlying storage system, or the path name is malformed.

D.6.2.3 void BiometricEvaluation::IO::Utility::copyDirectoryContents (const string & *sourcepath*, const string & *targetpath*, const bool *removesource* = false) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Copy the contents of a directory, optionally deleting the source directory contents when done.

Parameters

in	<i>sourcepath</i>	The name of the directory whose contents are to be moved.
in	<i>targetpath</i>	The name of the directory where the contents of the sourcepath are to be moved.
in	<i>removesource</i>	Flag indicating whether to remove the source directory after the copy is complete.

Exceptions

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

D.6.2.4 void BiometricEvaluation::IO::Utility::setAsideName (const string & name) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Set aside a file or directory name.

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

Parameters

in	path	The path name of the file or directory to be set aside.
----	------	---

Exceptions

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

D.6.2.5 uint64_t BiometricEvaluation::IO::Utility::getFileSize (const string & pathname) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Get the size of a file.

Parameters

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

Returns

The file size.

Exceptions

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

D.6.2.6 `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

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

D.6.2.7 `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.6.2.8 `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 true is returned; ambiguous when false is returned.

Returns

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

D.6.2.9 `int BiometricEvaluation::IO::Utility::makePath (const string & path, const mode_t mode)`

Create an entire directory tree.

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

Parameters

<i>in</i>	<i>path</i>	The path to create.
<i>in</i>	<i>mode</i>	The permission mode of each element in the path. See chmod(2).

Returns

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

D.6.2.10 `Memory::uint8Array BiometricEvaluation::IO::Utility::readFile (const string & path, ios_base::openmode mode = ios_base::binary) throw (Error::ObjectDoesNotExist, Error::StrategyError)`

Read the contents of a file into a buffer.

Parameters

<i>path</i>	Path to a file to be read.
<i>mode</i>	Bitwise OR'd arguments to send to the file stream constructor.

Returns

Contents of path in a buffer.

Exceptions

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

D.6.2.11 `void BiometricEvaluation::IO::Utility::writeFile (const uint8_t * data, const size_t size, const string & path, ios_base::openmode mode = ios_base::binary) throw (Error::ObjectExists, Error::StrategyError)`

Write the contents of a buffer to a file.

Parameters

<i>data</i>	Data buffer to write.
<i>size</i>	Size of data.
<i>path</i>	Path to file to create with contents of data.
<i>mode</i>	Bitwise OR'd arguments to send to the file stream constructor.

Exceptions

<i>ObjectExists</i>	path exists but truncate not set, or path exists and is a directory.
<i>StrategyError</i>	An error occurred when using the underlying storage system.

D.6.2.12 `void BiometricEvaluation::IO::Utility::writeFile (const Memory::uint8Array data, const string & path, ios_base::openmode mode = ios_base::binary) throw (Error::ObjectExists, Error::StrategyError)`

Write the contents of a buffer to a file.

Parameters

<i>data</i>	Data buffer to write.
<i>path</i>	Path to file to create with contents of data.
<i>mode</i>	Bitwise OR'd arguments to send to the file stream constructor.

Exceptions

<i>ObjectExists</i>	path exists but truncate not set, or path exists and is a directory.
<i>StrategyError</i>	An error occurred when using the underlying storage system.

D.6.2.13 `bool BiometricEvaluation::IO::Utility::isReadable (const string & pathname)`

Determine if a file can be opened with read permission.

Parameters

<i>in</i>	<i>pathname</i>	Path to the file to check.
-----------	-----------------	----------------------------

Returns

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

Note

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

See also

[BiometricEvaluation::IO::Utility::fileExists\(\)](#)

D.6.2.14 `bool BiometricEvaluation::IO::Utility::isWritable (const string & pathname)`

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

Parameters

in	<i>pathname</i>	Path to the file to check.
----	-----------------	----------------------------

Returns

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

Note

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

See also

[BiometricEvaluation::IO::Utility::fileExists\(\)](#)

D.6.2.15 `string BiometricEvaluation::IO::Utility::createTemporaryFile (const string & prefix = "", const string & parentDir = "/tmp") throw (Error::FileError, Error::MemoryError)`

Create a temporary file.

Parameters

in	<i>prefix</i>	String to be prefixed to the random temporary name.
in	<i>parentDir</i>	Where to place the temporary file.

Exceptions

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

Returns

Path to temporary file.

Note

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

D.6.2.16 `FILE* BiometricEvaluation::IO::Utility::createTemporaryFile (string & path, const string & prefix = "", const string & parentDir = "/tmp") throw (Error::FileError, Error::MemoryError)`

Create a temporary file.

Exclusivity to the file stream is guaranteed.

Parameters

out	<i>path</i>	Reference to a string that will hold the path to the opened temporary file.
in	<i>prefix</i>	String to be prefixed to the random temporary name.
in	<i>parentDir</i>	Where to place the temporary file.

Exceptions

<i>Error::FileError</i>	Could not create or close temporary file.
<i>Error::MemoryError</i>	Error allocating memory for file name.

Returns

Open file stream to path.

Note

Caller must `fclose(3)` the returned stream.

D.7 BiometricEvaluation::Memory Namespace Reference

Support for memory-related operations.

Classes

- class [AutoArray](#)
A C-style array wrapped in the facade of a C++ STL container.
- class [AutoBuffer](#)
- class [IndexedBuffer](#)
Manage a memory buffer with an index.

Typedefs

- typedef [AutoArray](#)< uint8_t > **uint8Array**
- typedef [AutoArray](#)< uint16_t > **uint16Array**
- typedef [AutoArray](#)< uint32_t > **uint32Array**

D.7.1 Detailed Description

Support for memory-related operations. The [Memory](#) package contains templates and classes that are used to manage memory, auto-sizing arrays, for example.

D.8 BiometricEvaluation::Process Namespace Reference

[Process](#) information and controls.

Classes

- class [ForkManager](#)
Manager implementation that starts Workers by calling fork(2).
- class [ForkWorkerController](#)
Wrapper of a Worker returned from a Process::ForkManager.
- class [Manager](#)
An interface for intranode process management classes.
- class [POSIXThreadManager](#)
Manager implementation that starts Workers in POSIX threads.
- class [POSIXThreadWorkerController](#)
Decorated Worker returned from a Process::POSIXThreadManager.
- class [Statistics](#)
The Statistics class provides an interface for gathering process statistics, such as memory usage, system time, etc.
- class [Worker](#)
An abstraction of an instance that performs work on given data.
- class [WorkerController](#)
Wrapper of a Worker returned from a Process::Manager.

Typedefs

- typedef map< string, tr1::shared_ptr< void > > [ParameterList](#)

D.8.1 Detailed Description

[Process](#) information and controls. The [Process](#) package gathers all process related matters, including a class to obtain resource usage statistics.

D.8.2 Typedef Documentation

D.8.2.1 typedef map< string, tr1::shared_ptr<void> > BiometricEvaluation::Process::ParameterList

Convenience typedef for parameter lists to child routines

D.9 BiometricEvaluation::System Namespace Reference

Operating system, hardware, etc.

Functions

- uint32_t [getCPUCount](#) () throw (Error::NotImplemented)
Obtain the number of central processing units that are online. Typically, this is the total CPU core count for the system.

- `uint64_t getRealMemorySize ()` throw (`Error::NotImplemented`)
Obtain the amount of real memory in the system.
- `double getLoadAverage ()` throw (`Error::NotImplemented`)
Obtain the system load average for the last minute.

D.9.1 Detailed Description

Operating system, hardware, etc. The `System` package gathers all system related matters, such as the operating system name, number of CPUs, etc.

D.9.2 Function Documentation

D.9.2.1 `uint32_t BiometricEvaluation::System::getCPUCount ()` throw (`Error::NotImplemented`)

Obtain the number of central processing units that are online. Typically, this is the total CPU core count for the system.

Returns

The number of processing units.

Exceptions

<i><code>Error::NotImplemented</code></i>	Not implemented for this operating system, or the underlying OS feature is not installed.
---	---

D.9.2.2 `uint64_t BiometricEvaluation::System::getRealMemorySize ()` throw (`Error::NotImplemented`)

Obtain the amount of real memory in the system.

Returns

The real memory size, in kilobytes.

Exceptions

<i><code>Error::NotImplemented</code></i>	Not implemented for this operating system, or the underlying OS feature is not installed.
---	---

D.9.2.3 `double BiometricEvaluation::System::getLoadAverage ()` throw (`Error::NotImplemented`)

Obtain the system load average for the last minute.

Returns

The system load average.

Exceptions

<i>Error::NotImplemented</i>	Not implemented for this operating system, or the underlying OS feature is not installed.
------------------------------	---

D.10 BiometricEvaluation::Text Namespace Reference

Text processing for string objects.

Functions

- void **removeLeadingTrailingWhitespace** (string &s)
Remove lead and trailing white space from a string object.
- string **digest** (const string &s, const string &digest="md5") throw (Error::StrategyError)
Compute the digest of a string.
- string **digest** (const void *buffer, const size_t buffer_size, const string &digest="md5") throw (Error::StrategyError)
Compute the digest of a memory buffer.
- vector< string > **split** (const string &str, const char delimiter, bool escape=true) throw (Error::ParameterError)
Return tokens bound by delimiters and the beginning and end of a string.
- string **filename** (const string &path)
Extract the filename portion of a pathname.
- string **dirname** (const string &path)
Extract the directory part of a pathname.

D.10.1 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.10.2 Function Documentation

D.10.2.1 string BiometricEvaluation::Text::digest (const string & s, const string & digest = "md5") throw (Error::StrategyError)

Compute the digest of a string.

Parameters

in	s	The string of which a digest should be computed.
in	digest	The digest to use. Any digest supported by OpenSSL is valid, and the default is MD5.

Returns

An ASCII representation of the hex digits composing the digest.

D.10.2.2 `string BiometricEvaluation::Text::digest (const void * buffer, const size_t buffer_size, const string & digest = "md5") throw (Error::StrategyError)`

Compute the digest of a memory buffer.

Parameters

in	<i>buffer</i>	The buffer of which a digest should be computed.
in	<i>buffer_size</i>	The size of buffer.
in	<i>digest</i>	The digest to use. Any digest supported by OpenSSL is valid, and the default is MD5.

Returns

An ASCII representation of the hex digits composing the digest.

D.10.2.3 `vector<string> BiometricEvaluation::Text::split (const string & str, const char delimiter, bool escape = true) throw (Error::ParameterError)`

Return tokens bound by delimiters and the beginning and end of a string.

Parameters

in	<i>str</i>	String to tokenize.
in	<i>delimiter</i>	Character that defines the end of a token. Any are valid, except '\ '.
in	<i>escape</i>	If the delimiter is prefixed with '\ ' in the string, do not split at that point and remove the '\ '.

Returns

Vector of string tokens, in order of appearance.

Note

If delimiter does not appear in string, the returned vector vector will still contain one item, str.

D.10.2.4 `string BiometricEvaluation::Text::filename (const string & path)`

Extract the filename portion of a pathname.

Parameters

in	<i>path</i>	Path from which to extract the filename portion.
----	-------------	--

Returns

Filename portion of path.

D.10.2.5 string BiometricEvaluation::Text::dirname (const string & path)

Extract the directory part of a pathname.

Parameters

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

Returns

Directory portion of path.

D.11 BiometricEvaluation::Time Namespace Reference

Support for time and timers.

Classes

- class [Timer](#)

This class can be used by applications to report the amount of time a block of code takes to execute.

- class [Watchdog](#)

A [Watchdog](#) object can be used by applications to limit the amount of processing time taken by a block of code.

Functions

- string [getCurrentTime](#) ()

Return the current time as a string.

- void **WatchdogSignalHandler** (int signo, siginfo_t *info, void *uap)

Variables

- const uint64_t **OneSecond** = 1000000
- const uint64_t **OneHalfSecond** = 500000
- const uint64_t **OneQuarterSecond** = 250000
- const uint64_t **OneEighthSecond** = 125000
- const int **MicrosecondsPerSecond** = 1000000
- const int **MillisecondsPerSecond** = 1000

D.11.1 Detailed Description

Support for time and timers. The [Time](#) package gathers all timing relating matters, such as Timers, [Watchdog](#) timers, etc. [Time](#) values are in microsecond units.

D.12 BiometricEvaluation::View Namespace Reference

[View](#) information.

Classes

- class [AN2KView](#)
A class to represent single biometric view and derived information.
- class [AN2KViewVariableResolution](#)
A class to represent single view based on an ANSI/NIST record.
- class [View](#)
A class to represent single biometric element view.

Functions

- `std::ostream & operator<< (std::ostream &stream, const AN2KView::DeviceMonitoringMode::Kind &kind)`
Output stream overload for DeviceMonitoringMode.
- `std::ostream & operator<< (std::ostream &stream, const AN2KViewVariableResolution::AN2KQualityMetric &qm)`
Output stream overload for AN2KQualityMetric.

D.12.1 Detailed Description

[View](#) information. The [View](#) package gathers all classes and other items that are related to a biometric view, which represents an image and all information derived from that image, such as fingerprint minutiae.

D.12.2 Function Documentation

D.12.2.1 `std::ostream& BiometricEvaluation::View::operator<< (std::ostream & stream, const AN2KViewVariableResolution::AN2KQualityMetric & qm)`

Output stream overload for AN2KQualityMetric.

Parameters

<code>in</code>	<code><i>stream</i></code>	Stream on which to append formatted AN2KQualityMetric information.
<code>in</code>	<code><i>qm</i></code>	AN2KQualityMetric information to append to stream.

Returns

stream with a qm textual representation appended.

Appendix E

Class Documentation

E.1 BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged Class Reference

Amputated or bandaged code.

```
#include <be_finger_an2kview_capture.h>
```

Public Types

- enum [Kind](#) { [Amputated](#), [Bandaged](#), [NA](#) }

E.1.1 Detailed Description

Amputated or bandaged code.

E.1.2 Member Enumeration Documentation

E.1.2.1 enum BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged::Kind

Enumerator:

Amputated Amputation

Bandaged Unable to print (e.g., bandaged)

NA Optional field -- not specified

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

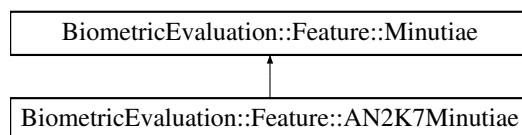
- `be_finger_an2kview_capture.h`

E.2 BiometricEvaluation::Feature::AN2K7Minutiae Class Reference

A class to represent a set of minutiae in an ANSI/NIST record.

```
#include <be_feature_an2k7minutiae.h>
```

Inheritance diagram for BiometricEvaluation::Feature::AN2K7Minutiae:



Classes

- class [EncodingMethod](#)
Methods for encoding minutiae data in an AN2K record.
- struct [FingerprintReadingSystem](#)
Representation of information about a fingerprint reader system.
- class [PatternClassification](#)
Pattern classification codes.

Public Types

- typedef std::vector < [PatternClassification::Entry](#) > **PatternClassificationSet**
- typedef struct [FingerprintReadingSystem](#) **FingerprintReadingSystem**

Public Member Functions

- [AN2K7Minutiae](#) (const std::string &filename, int recordNumber) throw (Error::DataError, Error::File-Error)
Construct an AN2K7 [Minutiae](#) object from file data.
- [AN2K7Minutiae](#) (Memory::uint8Array &buf, int recordNumber) throw (Error::DataError)
Construct an AN2K7 [Minutiae](#) object from data contained in a memory buffer.
- PatternClassificationSet [getPatternClassificationSet](#) () const
Obtain the set fingerprint pattern classifications.
- [FingerprintReadingSystem](#) [getOriginatingFingerprintReadingSystem](#) () const throw (Error::ObjectDoes-NotExist)
- MinutiaeFormat::Kind [getFormat](#) () const
Obtain the minutiae format kind.
- MinutiaPointSet [getMinutiaPoints](#) () const
Obtain the set of finger minutiae data points. The set may be empty.
- RidgeCountItemSet [getRidgeCountItems](#) () const
Obtain the set of ridge count data items. The set may be empty.
- CorePointSet [getCores](#) () const

- Obtains the set of core positions. The set may be empty.
- DeltaPointSet [getDeltas](#) () const
Obtains the set of delta positions. The set may be empty.

Static Public Member Functions

- static Finger::PatternClassification::Kind [convertPatternClassification](#) (const char *fpc) throw (Error::DataError)
Convert string read from AN2K record into a [PatternClassification](#).
- static Finger::PatternClassification::Kind [convertPatternClassification](#) (const [PatternClassification::Entry](#) &entry) throw (Error::DataError)
Convert a standard [PatternClassification::Entry](#) to a [PatternClassification::Kind](#).
- static EncodingMethod::Kind [convertEncodingMethod](#) (const char *mem) throw (Error::DataError)
Convert string read from AN2K record into a [EncodingMethod](#).
- static [Image::Coordinate](#) [convertCoordinate](#) (const char *str, bool calculateDistance=true) throw (Error::DataError)
Obtain a [Coordinate](#) given an AN2K entry.

E.2.1 Detailed Description

A class to represent a set of minutiae in an ANSI/NIST record.

Each minutiae point, ridge count item, core, and delta is represented in the native ANSI/NIST format.

E.2.2 Constructor & Destructor Documentation

E.2.2.1 BiometricEvaluation::Feature::AN2K7Minutiae::AN2K7Minutiae (const std::string & *filename*, int *recordNumber*) throw (Error::DataError, Error::FileError)

Construct an AN2K7 [Minutiae](#) object from file data.

The file contains a complete ANSI/NIST record, and an object of this class represents a single fingerprint minutiae record.

Parameters

in	<i>filename</i>	The name of the file containing the complete ANSI/NIST record.
in	<i>recordNumber</i>	Which fingerprint minutiae record to read from the complete AN2K record.

Exceptions

Error::FileError	An error occurred when opening or reading from the file.
Error::DataError	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.

E.2.2.2 BiometricEvaluation::Feature::AN2K7Minutiae::AN2K7Minutiae (Memory::uint8Array & *buf*, int *recordNumber*) throw (Error::DataError)

Construct an AN2K7 [Minutiae](#) object from data contained in a memory buffer.

The buffer contains a complete ANSI/NIST record, and an object of this class represents a single fingerprint minutiae record.

Parameters

in	<i>buf</i>	The memory buffer containing the complete ANSI/NIST record.
in	<i>recordNumber</i>	Which fingerprint minutiae record to read from the complete AN2K record.

Exceptions

Error::DataError	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.
----------------------------------	---

E.2.3 Member Function Documentation

E.2.3.1 static Finger::PatternClassification::Kind BiometricEvaluation::Feature::AN2K7Minutiae::convertPatternClassification (const char * *fpc*) throw (Error::DataError) [static]

Convert string read from AN2K record into a [PatternClassification](#).

Parameters

in	<i>fpc</i>	Value for pattern classification read from AN2K record.
----	------------	---

Exceptions

Error::DataError	Invalid value for <i>fpc</i> .
----------------------------------	--------------------------------

E.2.3.2 static Finger::PatternClassification::Kind BiometricEvaluation::Feature::AN2K7Minutiae::convertPatternClassification (const PatternClassification::Entry & *entry*) throw (Error::DataError) [static]

Convert a standard [PatternClassification::Entry](#) to a PatternClassification::Kind.

Parameters

in	<i>entry</i>	A standard pattern classification entry
----	--------------	---

Exceptions

Error::DataError	Non-standard pattern classification entry.
----------------------------------	--

E.2.3.3 `static EncodingMethod::Kind BiometricEvaluation::Feature::AN2K7Minutiae::convertEncodingMethod (const char * mem) throw (Error::DataError) [static]`

Convert string read from AN2K record into a [EncodingMethod](#).

Parameters

<i>in</i>	<i>mem</i>	Value for minutiae encoding method read from AN2K record.
-----------	------------	---

Exceptions

Error::DataError	Invalid value for mem.
----------------------------------	------------------------

E.2.3.4 `PatternClassificationSet BiometricEvaluation::Feature::AN2K7Minutiae::getPatternClassificationSet () const`

Obtain the set fingerprint pattern classifications.

The code returned may be a standard code or user-defined. Applications should call `isPatternClassificationStandard()` to check.

E.2.3.5 `FingerprintReadingSystem BiometricEvaluation::Feature::AN2K7Minutiae::getOriginatingFingerprintReadingSystem () const throw (Error::ObjectDoesNotExist)`

Obtain the originating fingerprint reading system.

Exceptions

Error::ObjectDoesNotExist	The optional OFR field has been excluded.
---	---

E.2.3.6 `static Image::Coordinate BiometricEvaluation::Feature::AN2K7Minutiae::convertCoordinate (const char * str, bool calculateDistance = true) throw (Error::DataError) [static]`

Obtain a Coordinate given an AN2K entry.

This AN2K entry is formatted as "XXXXYYYY".

Parameters

<i>in</i>	<i>str</i>	Coordinate string from an AN2K record.
<i>in</i>	<i>calculateDistance</i>	Whether or not to calculate the [xy]Distance portion of the Coordinate.

Returns

[Image::Coordinate](#) representation of str.

Exceptions

Error::DataError	Invalid format of str.
----------------------------------	------------------------

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

- `be_feature_an2k7minutiae.h`

E.3 BiometricEvaluation::Finger::AN2KMinutiaeDataRecord Class Reference

Representation of a Type-9 Record from an AN2K file.

```
#include <be_finger_an2kminutiae_data_record.h>
```

Public Member Functions

- [AN2KMinutiaeDataRecord](#) (const string &filename, int recordNumber) throw (Error::DataError, Error::FileError)
Construct an [AN2KMinutiaeDataRecord](#) object from data contained in a file on disk.
- [AN2KMinutiaeDataRecord](#) (Memory::uint8Array &buf, int recordNumber) throw (Error::DataError)
Construct an [AN2KMinutiaeDataRecord](#) object from data contained in a memory buffer.
- `tr1::shared_ptr < Feature::AN2K7Minutiae > getAN2K7Minutiae () const`
Obtain the "standard" minutiae data from this Type-9 Record (fields 9.005 - 9.012).
- `Impression::Kind getImpressionType () const`
Return impression type field from Type-9 Record.
- `map< uint16_t, Memory::uint8Array > getRegisteredVendorBlock` (Feature::MinutiaeFormat::Kind vendor) const throw (Error::NotImplemented)
Obtain data recorded in a registered vendor minutiae block found in this Type-9 Record.

E.3.1 Detailed Description

Representation of a Type-9 Record from an AN2K file.

Type-9 Records may contain only "standard" minutiae data (fields 9.005 - 9.012) or any combination of "standard" minutiae data and registered vendor minutiae data (several vendors from fields 9.013 - 9.175).

E.3.2 Constructor & Destructor Documentation

E.3.2.1 BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::AN2KMinutiaeDataRecord (const string & filename, int recordNumber) throw (Error::DataError, Error::FileError)

Construct an [AN2KMinutiaeDataRecord](#) object from data contained in a file on disk.

The file contains a complete ANSI/NIST record, and an object of this class represents a single fingerprint minutiae record.

Parameters

in	<i>filename</i>	The name of the file containing the complete ANSI/NIST record.
in	<i>recordNumber</i>	Which fingerprint minutiae record to read from the complete AN2K record.

Exceptions

<i>Error::FileError</i>	An error occurred when opening or reading from the file.
<i>Error::DataError</i>	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.

E.3.2.2 BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::AN2KMinutiaeDataRecord (Memory::uint8Array & *buf*, int *recordNumber*) throw (Error::DataError)

Construct an [AN2KMinutiaeDataRecord](#) object from data contained in a memory buffer.

The buffer contains a complete ANSI/NIST record, and an object of this class represents a single fingerprint minutiae record.

Parameters

in	<i>buf</i>	The memory buffer containing the complete ANSI/NIST record.
in	<i>recordNumber</i>	Which fingerprint minutiae record to read from the complete AN2K record.

Exceptions

<i>Error::DataError</i>	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.
---	---

E.3.3 Member Function Documentation

E.3.3.1 tr1::shared_ptr<Feature::AN2K7Minutiae> BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getAN2K7Minutiae () const

Obtain the "standard" minutiae data from this Type-9 Record (fields 9.005 - 9.012).

Returns

Shared pointer to an AN2KMinutiae object containing the standard format minutiae data found in this Type-9 Record.

E.3.3.2 Impression::Kind BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getImpressionType () const

Return impression type field from Type-9 Record.

Returns

[Impression](#) type of the image from which minutiae points were generated.

E.3.3.3 `map<uint16_t, Memory::uint8Array> BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getRegisteredVendorBlock (Feature::MinutiaeFormat::Kind vendor) const throw (Error::NotImplemented)`

Obtain data recorded in a registered vendor minutiae block found in this Type-9 Record.

Parameters

<code>in</code>	<code><i>vendor</i></code>	The vendor whose registered minutiae blocks are being requested.
-----------------	----------------------------	--

Returns

A map of the registered vendor minutiae block fields. The map key is the AN2K Field number. The value is a uint8Array of the ASCII data found at that field. All Fields will be present as keys even if there was no data recorded in that Field.

Exceptions

<i>Error::NotImplemented</i>	Cannot return a map of fields for vendor, likely because there exists a better, native implementation of accessing minutiae data in AN2KMinutiaeDataRecord .
------------------------------	--

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

- `be_finger_an2kminutiae_data_record.h`

E.4 BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric Struct Reference

A structure to represent an AN2K quality metric.

```
#include <be_view_an2kview_varres.h>
```

Public Attributes

- `Finger::Position::Kind` **position**
- `uint8_t` **score**
- `uint16_t` **vendorID**
- `uint16_t` **productCode**

E.4.1 Detailed Description

A structure to represent an AN2K quality metric.

The quality metric is an optional field in the Type-13 (Latent), Type-14 (Fingerprint and Segmentation) and Type-15 (Palmprint). The NIST Quality Metric is also returned via this structure.

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

- `be_view_an2kview_varres.h`

E.5 BiometricEvaluation::DataInterchange::AN2KRecord Class Reference

A class to represent an entire ANSI/NIST record.

```
#include <be_data_interchange_an2k.h>
```

Classes

- struct [CharacterSet](#)
- struct [DomainName](#)

Representation of a domain name for the user-defined Type-2 logical record implementation.

Public Types

- typedef struct [DomainName](#) [DomainName](#)
- typedef struct [CharacterSet](#) [CharacterSet](#)

Public Member Functions

- [AN2KRecord](#) (const std::string filename) throw ([Error::FileError](#), [Error::DataError](#))
Constructor taking an AN2K record from a file.
- [AN2KRecord](#) ([Memory::uint8Array](#) &buf) throw ([Error::DataError](#))
Constructor taking an AN2K record from a buffer.
- string [getVersionNumber](#) () const
Obtain the AN2K record version.
- string [getDate](#) () const
Obtain the AN2K record date.
- string [getDestinationAgency](#) () const
Obtain the destination agency ID.
- string [getOriginatingAgency](#) () const
Obtain the originating agency ID.
- string [getTransactionControlNumber](#) () const
Obtain the transcantion control number.
- string [getNativeScanningResolution](#) () const
Obtain the native scanning resolution.
- string [getNominalTransmittingResolution](#) () const
Obtain the nominal transmitting resolution.
- uint32_t [getFingerLatentCount](#) () const
Obtain the count of latent (Type-13) finger views.
- std::vector < [Finger::AN2KViewLatent](#) > [getFingerLatents](#) () const
Obtain all latent (Type-13) finger views.
- uint32_t [getFingerCaptureCount](#) () const
Obtain the count of capture (Type-14) finger views.
- std::vector < [Finger::AN2KViewCapture](#) > [getFingerCaptures](#) () const
Obtain all capture (Type-14) finger views.

- `std::vector < Finger::AN2KMinutiaeDataRecord > getMinutiaeDataRecordSet () const`
Obtain all minutiae (Type-9) data.
- `uint8_t getPriority () const`
Obtain the urgency with which a response is required.
- `DomainName getDomainName () const`
Obtain the identifier of the domain name for the user-defined Type-2 logical record implementation.
- `struct tm getGreenwichMeanTime () const`
Obtain the date and time of encoding in terms of GMT units.
- `std::vector< CharacterSet > getDirectoryOfCharacterSets () const`
Obtain the list of character sets other than 7-bit ASCII that may appear in the transaction.

Static Public Member Functions

- `static set< int > recordLocations (Memory::uint8Array &buf, const View::AN2KView::RecordType::Kind recordType) throw (Error::DataError)`
Find the position within a buffer of all Records of a particular type.
- `static set< int > recordLocations (const ANSI_NIST *an2k, const View::AN2KView::RecordType::Kind recordType)`
Find the position within an ANSI_NIST struct of all Records of a particular type.

E.5.1 Detailed Description

A class to represent an entire ANSI/NIST record.

An object of this class can be used to retrieve all the general record information, finger views, and other components of the ANSI/NIST record.

E.5.2 Member Typedef Documentation

E.5.2.1 `typedef struct DomainName BiometricEvaluation::DataInterchange::AN2KRecord::Domain-Name`

Convenience typedef for struct [DomainName](#)

E.5.2.2 `typedef struct CharacterSet BiometricEvaluation::DataInterchange::AN2KRecord::Character-Set`

Convenience typedef for struct [CharacterSet](#)

E.5.3 Constructor & Destructor Documentation

E.5.3.1 `BiometricEvaluation::DataInterchange::AN2KRecord::AN2KRecord (const std::string filename) throw (Error::FileError, Error::DataError)`

Constructor taking an AN2K record from a file.

Parameters

in	<i>filename</i>	The name of the file containing the complete ANSI/NIST record.
----	-----------------	--

Exceptions

<i>Error::FileError</i>	An error occurred when opening or reading the file.
<i>Error::DataError</i>	An error occurred when processing the AN2K record.

E.5.3.2 BiometricEvaluation::DataInterchange::AN2KRecord::AN2KRecord (Memory::uint8Array & *buf*) throw (Error::DataError)

Constructor taking an AN2K record from a buffer.

Parameters

in	<i>buf</i>	The memory buffer containing the complete ANSI/NIST record.
----	------------	---

Exceptions

<i>Error::DataError</i>	An error occurred when processing the AN2K record.
---	--

E.5.4 Member Function Documentation

E.5.4.1 static set<int> BiometricEvaluation::DataInterchange::AN2KRecord::recordLocations (Memory::uint8Array & *buf*, const View::AN2KView::RecordType::Kind *recordType*) throw (Error::DataError) [static]

Find the position within a buffer of all Records of a particular type.

Parameters

in	<i>buf</i>	AN2K Buffer to search.
in	<i>recordType</i>	The ID of the Record to search for.

Returns

Set of integer positions within *buf* where a *recordType* Record is located.

Exceptions

<i>Error::DataError</i>	An error occurred when processing the AN2K record.
---	--

E.5.4.2 static set<int> BiometricEvaluation::DataInterchange::AN2KRecord::recordLocations (const ANSI_NIST * *an2k*, const View::AN2KView::RecordType::Kind *recordType*) [static]

Find the position within an ANSI_NIST struct of all Records of a particular type.

Parameters

in	<i>an2k</i>	ANSI_NIST struct to search.
in	<i>recordType</i>	The ID of the Record to search for.

Returns

Set of integer positions within the ANSI_NIST struct where a recordType Record is located.

E.5.4.3 string BiometricEvaluation::DataInterchange::AN2KRecord::getVersionNumber () const

Obtain the AN2K record version.

Returns

The record version field in the Type-1 record.

E.5.4.4 string BiometricEvaluation::DataInterchange::AN2KRecord::getDate () const

Obtain the AN2K record date.

Returns

The date field in the Type-1 record.

E.5.4.5 string BiometricEvaluation::DataInterchange::AN2KRecord::getDestinationAgency () const

Obtain the destination agency ID.

Returns**E.5.4.6 string BiometricEvaluation::DataInterchange::AN2KRecord::getOriginatingAgency () const**

Obtain the originating agency ID.

Returns**E.5.4.7 string BiometricEvaluation::DataInterchange::AN2KRecord::getTransactionControlNumber () const**

Obtain the transaction control number.

Returns

E.5.4.8 `string BiometricEvaluation::DataInterchange::AN2KRecord::getNativeScanningResolution () const`

Obtain the native scanning resolution.

Returns**E.5.4.9** `string BiometricEvaluation::DataInterchange::AN2KRecord::getNominalTransmittingResolution () const`

Obtain the nominal transmitting resolution.

Returns**E.5.4.10** `uint32_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerLatentCount () const`

Obtain the count of latent (Type-13) finger views.

Returns

The number of latent finger views in the AN2K record.

E.5.4.11 `std::vector<Finger::AN2KViewLatent> BiometricEvaluation::DataInterchange::AN2KRecord::getFingerLatents () const`

Obtain all latent (Type-13) finger views.

The returned vector will be empty when no latent views are present in the [AN2KRecord](#).

Returns

A vector of AN2KViewLatent objects, each representing a single latent finger view.

E.5.4.12 `uint32_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerCaptureCount () const`

Obtain the count of capture (Type-14) finger views.

Returns

The number of latent finger views in the AN2K record.

E.5.4.13 `std::vector<Finger::AN2KViewCapture> BiometricEvaluation::DataInterchange::AN2KRecord::getFingerCaptures () const`

Obtain all capture (Type-14) finger views.

The returned vector will be empty when no capture views are present in the [AN2KRecord](#).

Returns

A vector of AN2KViewCapture objects, each representing a single capture finger view.

E.5.4.14 `std::vector<Finger::AN2KMinutiaeDataRecord> BiometricEvaluation::DataInterchange::AN2KRecord::getMinutiaeDataRecordSet () const`

Obtain all minutiae (Type-9) data.

Returns

A vector of AN2KMinutiaeDataRecord objects, each representing a single Type-9 Record.

E.5.4.15 `uint8_t BiometricEvaluation::DataInterchange::AN2KRecord::getPriority () const`

Obtain the urgency with which a response is required.

Returns

Priority (1:High - 9:Low)

E.5.4.16 `DomainName BiometricEvaluation::DataInterchange::AN2KRecord::getDomainName () const`

Obtain the identifier of the domain name for the user-defined Type-2 logical record implementation.

Returns

[DomainName](#) struct with identifier and version information (if defined).

E.5.4.17 `struct tm BiometricEvaluation::DataInterchange::AN2KRecord::getGreenwichMeanTime () const`
[read]

Obtain the date and time of encoding in terms of GMT units.

Returns

struct tm encoding of the GMT field.

E.5.4.18 `std::vector<CharacterSet> BiometricEvaluation::DataInterchange::AN2KRecord::getDirectoryOfCharacterSets () const`

Obtain the list of character sets other than 7-bit ASCII that may appear in the transaction.

Returns

Vector of [CharacterSet](#) structs representing other character sets that may appear in the transaction.

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

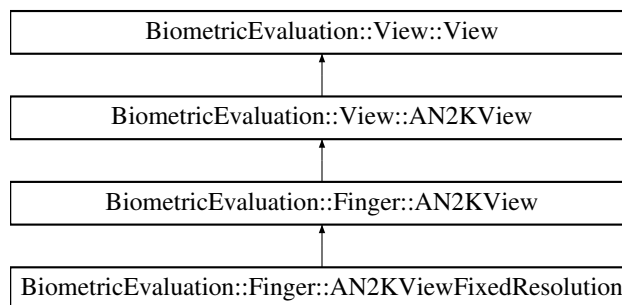
- `be_data_interchange_an2k.h`

E.6 BiometricEvaluation::Finger::AN2KView Class Reference

A class to represent single finger view and derived information.

```
#include <be_finger_an2kview.h>
```

Inheritance diagram for BiometricEvaluation::Finger::AN2KView:



Public Member Functions

- `vector< AN2KMinutiaeDataRecord > getMinutiaeDataRecordSet () const` throw (Error::DataError)
Obtain the set of minutiae records.
- `Finger::PositionSet getPositionSet () const`
Obtain the set of finger positions.
- `Finger::Impression::Kind getImpressionType () const`
Obtain the finger impression code.

Static Public Member Functions

- `static Finger::Position::Kind convertPosition (int an2kFGP) throw (Error::DataError)`
Convert a compression algorithm indicator from an AN2K finger image record.
- `static Finger::PositionSet populateFGP (FIELD *field) throw (Error::DataError)`
Read the finger positions from an AN2K record.

- static `Finger::Impression::Kind` [convertImpression](#) (const unsigned char *str) throw (Error::DataError)
Convert an impression code from a string.
- static `Finger::FingerImageCode::Kind` [convertFingerImageCode](#) (const char *str) throw (Error::DataError)
Convert an finger image code from a string.

Protected Member Functions

- [AN2KView](#) (const std::string filename, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)
Construct an AN2K finger view from a file.
- [AN2KView](#) ([Memory::uint8Array](#) &buf, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)
- void [addMinutiaeDataRecord](#) ([Finger::AN2KMinutiaeDataRecord](#) &mdr)
Mutator for the [AN2KMinutiaeDataRecord](#) set.
- void [setPosition](#) ([Finger::PositionSet](#) &ps)
Mutator for the position set.
- void [setImpressionType](#) ([Finger::Impression::Kind](#) &imp)
Mutator for the impression type.

E.6.1 Detailed Description

A class to represent single finger view and derived information.

A base [Finger::AN2KView](#) object represents an ANSI/NIST Type-3/4/5/6 record, and can return the image as well as the other information associated with that image, such as the minutiae from the corresponding Type-9 record.

For these types of records, the image resolution and scan resolution are identical. For compressed images, applications can compare the image resolution and size taken from the Type-3/4/5/6 record to that returned by the [Image](#) object directly.

E.6.2 Constructor & Destructor Documentation

- E.6.2.1** `BiometricEvaluation::Finger::AN2KView::AN2KView (const std::string filename, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError) [protected]`

Construct an AN2K finger view from a file.

The file must contain the entire AN2K record, not just the finger image and/or minutiae records.

E.6.3 Member Function Documentation

- E.6.3.1** `static Finger::Position::Kind BiometricEvaluation::Finger::AN2KView::convertPosition (int an2kFGP) throw (Error::DataError) [static]`

Convert a compression algorithm indicator from an AN2K finger image record.

Parameters

in	<i>an2kFGP</i>	A finger position code as defined by the AN2K standard.
----	----------------	---

Exceptions

Error::DataError	The position code is invalid.
----------------------------------	-------------------------------

E.6.3.2 `static Finger::PositionSet BiometricEvaluation::Finger::AN2KView::populateFGP (FIELD * field) throw (Error::DataError) [static]`

Read the finger positions from an AN2K record.

An AN2K finger image record can have multiple values * for the finger position. Pull them out of the position field and return them as a set.

Exceptions

Error::DataError	The data contains an invalid value.
----------------------------------	-------------------------------------

E.6.3.3 `static Finger::FingerImageCode::Kind BiometricEvaluation::Finger::AN2KView::convertFingerImageCode (const char * str) throw (Error::DataError) [static]`

Convert an finger image code from a string.

Parameters

in	<i>str</i>	The character string containing the image code.
----	------------	---

Returns

A [FingerImageCode](#) value.

Exceptions

Error::DataError	The string contains an invalid image code.
----------------------------------	--

E.6.3.4 `vector<AN2KMinutiaeDataRecord> BiometricEvaluation::Finger::AN2KView::getMinutiaeDataRecordSet () const throw (Error::DataError)`

Obtain the set of minutiae records.

Because it is possible to have more than one Type-9 record associated with a finger view, this method returns a set of objects, each one representing a single Type-9 record.

Reimplemented from [BiometricEvaluation::View::AN2KView](#).

E.6.3.5 Finger::PositionSet BiometricEvaluation::Finger::AN2KView::getPositions () const

Obtain the set of finger positions.

An AN2K finger image record contains a set of possible finger positions. This method returns that set as read from the image record. Any minutiae record (Type-9) associated with this image will have its own set of positions.

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

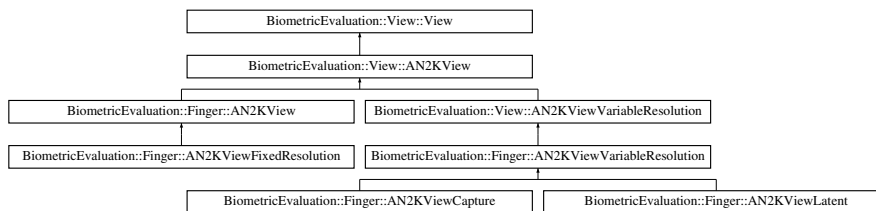
- be_finger_an2kview.h

E.7 BiometricEvaluation::View::AN2KView Class Reference

A class to represent single biometric view and derived information.

```
#include <be_view_an2kview.h>
```

Inheritance diagram for BiometricEvaluation::View::AN2KView:



Classes

- class [DeviceMonitoringMode](#)
The level of human monitoring for the image capture device.
- class [RecordType](#)
The type of AN2K record.

Public Member Functions

- [AN2KView](#) (const std::string filename, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)
Construct an AN2K view from a file.
- [AN2KView](#) (Memory::uint8Array &buf, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)
Construct an AN2K view from a buffer.
- tr1::shared_ptr< [Image::Image](#) > [getImage](#) () const
Obtain the image used for the finger view.
- [Image::Size](#) [getImageSize](#) () const
Obtain the image size.
- [Image::Resolution](#) [getImageResolution](#) () const

- Obtain the image resolution.*
 - `uint32_t getImageDepth () const`
- Obtain the image depth.*
 - `Image::CompressionAlgorithm::Kind getCompressionAlgorithm () const`
- Obtain the compression algorithm used on the image.*
 - `Image::Resolution getScanResolution () const`
- Obtain the image scan resolution.*
 - `vector < Finger::AN2KMinutiaeDataRecord > getMinutiaeDataRecordSet () const throw (Error::DataError)`
- Obtain the set of minutiae records.*
 - `RecordType::Kind getRecordType () const`
- Obtain the ANSI-NIST record type.*

Static Public Member Functions

- static `DeviceMonitoringMode::Kind convertDeviceMonitoringMode (const char *dmm) throw (Error::DataError)`
- Convert a device monitoring mode indicator from an AN2K record.*
- static `Image::CompressionAlgorithm::Kind convertCompressionAlgorithm (const uint16_t recordType, const unsigned char *an2kValue) throw (Error::ParameterError, Error::DataError)`

Static Public Attributes

- static const double `MinimumScanResolutionPPMM`
- Constants to define the minimum resolution used for fingerprint images in an AN2k record.*
- static const double `HalfMinimumScanResolutionPPMM`
- static const int `FixedResolutionBitDepth = 8`
- The defined bit-depth for fixed-resolution images.*

Protected Member Functions

- `Memory::AutoBuffer< ANSI_NIST > getAN2K () const`
- Obtain the complete ANSI/NIST record set.*
- `RECORD * getAN2KRecord () const`
- Obtain a pointer to the single ANSI/NIST record.*
- void `setImageData (const Memory::AutoArray< uint8_t > &imageData)`
- Mutator for the image data.*
- void `setImageResolution (const Image::Resolution &ir)`
- Mutator for the image resolution.*
- void `setImageDepth (const uint32_t depth)`
- Mutator for the image depth.*
- void `setScanResolution (const Image::Resolution &ir)`
- Mutator for the scan resolution.*
- void `setCompressionAlgorithm (const Image::CompressionAlgorithm::Kind &ca)`
- Mutator for the compression algorithm.*

E.7.1 Detailed Description

A class to represent single biometric view and derived information.

This abstraction represents the image and derived information taken from an ANSI/NIST record.

For these types of records, the image resolution and scan resolution are identical. For compressed images, applications can compare the image resolution and size taken from the Type-3/4/5/6 record to that returned by the [Image](#) object directly.

E.7.2 Constructor & Destructor Documentation

E.7.2.1 `BiometricEvaluation::View::AN2KView::AN2KView (const std::string filename, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)`

Construct an AN2K view from a file.

The file must contain the entire AN2K record, not just the image and other view-related records.

E.7.2.2 `BiometricEvaluation::View::AN2KView::AN2KView (Memory::uint8Array & buf, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)`

Construct an AN2K view from a buffer.

The buffer must contain the entire AN2K record, not just the image and other view-related records.

E.7.3 Member Function Documentation

E.7.3.1 `static DeviceMonitoringMode::Kind BiometricEvaluation::View::AN2KView-
::convertDeviceMonitoringMode (const char * dmm) throw (Error::DataError)
[static]`

Convert a device monitoring mode indicator from an AN2K record.

Parameters

<i>dmm</i>	Item value for device monitoring mode from an AN2K record.
------------	--

Returns

[DeviceMonitoringMode](#) representation of dmm.

Exceptions

Error::DataError	Invalid format of dmm.
----------------------------------	------------------------

E.7.3.2 `static Image::CompressionAlgorithm::Kind BiometricEvaluation::View::AN2KView::convert-CompressionAlgorithm (const uint16_t recordType, const unsigned char * an2kValue) throw (Error::ParameterError, Error::DataError) [static]`

Convert a compression algorithm indicator from an AN2K finger image record.

Parameters

<i>recordType</i>	The AN2K record type as an integer, allowing the value taken directly from the AN2K record or a RecordType::Kind to be passed in.
<i>an2kValue</i>	Compression type data as read from an AN2K record.

Returns

The compression algorithm.

Exceptions

<i>Error::DataError</i>	Invalid compression algorithm for record type.
<i>Error::ParameterError</i>	Invalid record type.

E.7.3.3 `tr1::shared_ptr<Image::Image> BiometricEvaluation::View::AN2KView::getImage () const [virtual]`

Obtain the image used for the finger view.

Not all finger views will have an image, however the derived information, such as minutiae, may be present.

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

E.7.3.4 `Image::Size BiometricEvaluation::View::AN2KView::getImageSize () const [virtual]`

Obtain the image size.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image size must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

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

E.7.3.5 `Image::Resolution BiometricEvaluation::View::AN2KView::getImageResolution () const [virtual]`

Obtain the image resolution.

[Image](#) resolution is taken from the biometric record, and not from the image data. In some cases, the resolution may be the components of the pixel ratio, and applications must check the [Image::Resolution::units](#) field for value NA.

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

E.7.3.6 `uint32_t BiometricEvaluation::View::AN2KView::getImageDepth () const` [virtual]

Obtain the image depth.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image depth must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

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

E.7.3.7 `Image::CompressionAlgorithm::Kind BiometricEvaluation::View::AN2KView::getCompressionAlgorithm () const` [virtual]

Obtain the compression algorithm used on the image.

This value is as present in the biometric record, and not obtained from the image data itself.

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

E.7.3.8 `Image::Resolution BiometricEvaluation::View::AN2KView::getScanResolution () const` [virtual]

Obtain the image scan resolution.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image resolution must be equal, but applications can check for inconsistencies.

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

E.7.3.9 `vector<Finger::AN2KMinutiaeDataRecord> BiometricEvaluation::View::AN2KView::getMinutiaeDataRecordSet () const` throw (Error::DataError)

Obtain the set of minutiae records.

Each [AN2KViewVariableResolution](#) may have more than one associated Type-9 record and each Type-9 record may have more than one minutiae format.

Returns

A vector of minutiae data records.

Reimplemented in [BiometricEvaluation::Finger::AN2KView](#).

E.7.3.10 `RecordType::Kind BiometricEvaluation::View::AN2KView::getRecordType () const`

Obtain the ANSI-NIST record type.

Returns

The type of record used to construct this object.

E.7.3.11 RECORD* BiometricEvaluation::View::AN2KView::getAN2KRecord () const [protected]

Obtain a pointer to the single ANSI/NIST record.

Child classes use this method to obtain a pointer to the specific ANSI/NIST record that was searched for by this class object.

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

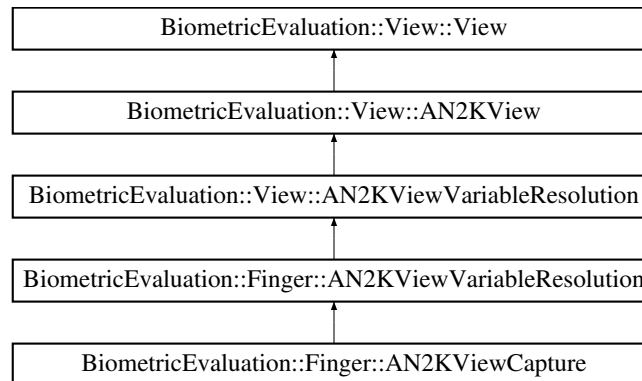
- be_view_an2kview.h

E.8 BiometricEvaluation::Finger::AN2KViewCapture Class Reference

Represents an ANSI/NIST variable-resolution finger image.

```
#include <be_finger_an2kview_capture.h>
```

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewCapture:

**Classes**

- class [AmputatedBandaged](#)
Amputated or bandaged code.
- struct [FingerSegmentPosition](#)
Locations of an individual finger segment in a slap.

Public Types

- typedef struct [FingerSegmentPosition](#) **FingerSegmentPosition**
- typedef std::vector < [FingerSegmentPosition](#) > **FingerSegmentPositionSet**

Public Member Functions

- [AN2KViewCapture](#) (const std::string &filename, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

- Construct an AN2K finger view from a file.*
- [AN2KViewCapture](#) ([Memory::uint8Array](#) &buf, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)
 - Construct an AN2K finger view using from a memory buffer.*
- [QualityMetricSet](#) [extractNISTQuality](#) (const FIELD *field) throw (Error::DataError)
 - Extract the NQM information from an AN2K FIELD.*
- [PositionDescriptors](#) [getPrintPositionDescriptors](#) () const
 - Return search position descriptors.*
- [QualityMetricSet](#) [getNISTQualityMetric](#) () const
 - Obtain the NIST quality metric for all segmented finger images.*
- [QualityMetricSet](#) [getSegmentationQualityMetric](#) () const
 - Obtain the segmentation quality metric for all segmented finger images.*
- [AmputatedBandaged::Kind](#) [getAmputatedBandaged](#) () const
- [FingerSegmentPositionSet](#) [getFingerSegmentPositionSet](#) () const
- [FingerSegmentPositionSet](#) [getAlternateFingerSegmentPositionSet](#) () const
- [QualityMetricSet](#) [getFingerprintQualityMetric](#) () const
 - Obtain metrics for fingerprint image quality score data for the image stored in this record.*

Static Public Member Functions

- static [AmputatedBandaged::Kind](#) [convertAmputatedBandaged](#) (const char *ampcd) throw (Error::DataError)
 - Convert string read from AN2K record into a [AmputatedBandaged](#) code.*
- static [FingerSegmentPosition](#) [convertFingerSegmentPosition](#) (const SUBFIELD *sf) throw (Error::DataError)
 - Convert SUBFIELD read from AN2K record into a [FingerSegmentPosition](#) struct.*
- static [FingerSegmentPosition](#) [convertAlternateFingerSegmentPosition](#) (const SUBFIELD *sf) throw (Error::DataError)
 - Convert SUBFIELD read from AN2K record into an [AlternateFingerSegmentPosition](#) struct.*

E.8.1 Detailed Description

Represents an ANSI/NIST variable-resolution finger image.

If the complete ANSI/NIST record contains a corresponding Type-9 (finger minutiae) record, an object of this class can be used to retrieve the minutiae set(s).

E.8.2 Constructor & Destructor Documentation

- #### E.8.2.1 [BiometricEvaluation::Finger::AN2KViewCapture::AN2KViewCapture](#) (const std::string & filename, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

Construct an AN2K finger view from a file.

The file must contain the entire AN2K record, not just the finger image and/or minutiae records. The object is constructed based on the nth variable resolution record found.

Parameters

in	<i>filename</i>	The name of the file containing the complete ANSI/NIST record.
in	<i>recordNumber</i>	The number of variable resolution record to read from the complete AN2K record.

Exceptions

<i>Error::ParameterError</i>	
<i>Error::DataError</i>	
<i>Error::FileError</i>	An error occurred when opening or reading the file.

E.8.2.2 BiometricEvaluation::Finger::AN2KViewCapture (Memory::uint8Array & *buf*, const uint32_t *recordNumber*) throw (Error::ParameterError, Error::DataError)

Construct an AN2K finger view using from a memory buffer.

The buffer must contain the entire AN2K record, not just the finger image and/or minutiae records.

E.8.3 Member Function Documentation

E.8.3.1 static AmputatedBandaged::Kind BiometricEvaluation::Finger::AN2KViewCapture::convertAmputatedBandaged (const char * *ampcd*) throw (Error::DataError) [static]

Convert string read from AN2K record into a [AmputatedBandaged](#) code.

Parameters

in	<i>ampcd</i>	Value for amputated bandaged code read from an AN2K record.
----	--------------	---

Exceptions

<i>Error::DataError</i>	Invalid value for <i>ampcd</i> .
---	----------------------------------

E.8.3.2 static FingerSegmentPosition BiometricEvaluation::Finger::AN2KViewCapture::convertFingerSegmentPosition (const SUBFIELD * *sf*) throw (Error::DataError) [static]

Convert SUBFIELD read from AN2K record into a [FingerSegmentPosition](#) struct.

Parameters

in	<i>sf</i>	Subfield value for a single finger segment position read from an AN2K record.
----	-----------	---

Exceptions

<i>Error::DataError</i>	Invalid value within sf.
---	--------------------------

E.8.3.3 `static FingerSegmentPosition BiometricEvaluation::Finger::AN2KViewCapture::convert-AlternateFingerSegmentPosition (const SUBFIELD * sf) throw (Error::DataError)`
`[static]`

Convert SUBFIELD read from AN2K record into an AlternateFingerSegmentPosition struct.

Parameters

<code>in</code>	<code><i>sf</i></code>	Subfield value for a single alternate finger segment position read from an AN2K record.
-----------------	------------------------	---

Exceptions

<i>Error::DataError</i>	Invalid value with sf.
---	------------------------

E.8.3.4 `QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::extractNISTQuality (const FIELD * field) throw (Error::DataError)`

Extract the NQM information from an AN2K FIELD.

Parameters

<code><i>field</i></code>	FIELD containing properly formatted NQM data
---------------------------	--

Returns

QualityMetricSet representation of field.

Exceptions

<i>Error::DataError</i>	Invalid format of field for NQM.
---	----------------------------------

E.8.3.5 `QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::getNISTQualityMetric () const`

Obtain the NIST quality metric for all segmented finger images.

Returns

QualityMetricSet containing the NIST quality metric for all segmented finger images.

Vendor ID and Product Code are undefined, as they are unused by NQM.

E.8.3.6 QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::getSegmentationQualityMetric () const

Obtain the segmentation quality metric for all segmented finger images.

Returns

QualityMetricSet containing the segmentation quality metric for all segmented finger images.

E.8.3.7 AmputatedBandaged::Kind BiometricEvaluation::Finger::AN2KViewCapture::getAmputated-Bandaged () const**Returns**

Optional amputated or bandaged code.

E.8.3.8 FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getFingerSegment-PositionSet () const**Returns**

Optional set of rectangular finger segment positions for all finger segments.

E.8.3.9 FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getAlternateFinger-SegmentPositionSet () const**Returns**

Optional set of polygonal finger segment positions for all finger segments.

E.8.3.10 QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::getFingerprintQualityMetric () const

Obtain metrics for fingerprint image quality score data for the image stored in this record.

Returns

Fingerprint quality metrics

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

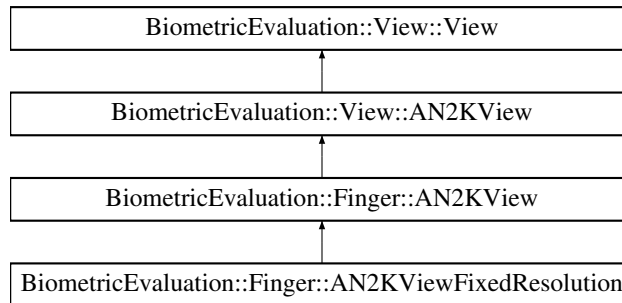
- be_finger_an2kview_capture.h

E.9 BiometricEvaluation::Finger::AN2KViewFixedResolution Class Reference

A class to represent single finger view and derived information.

```
#include <be_finger_an2kview_fixedres.h>
```

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewFixedResolution:



Public Member Functions

- [AN2KViewFixedResolution](#) (const std::string filename, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)
Construct an AN2K finger view from a file.
- [AN2KViewFixedResolution](#) (Memory::uint8Array &buf, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

E.9.1 Detailed Description

A class to represent single finger view and derived information.

A base [Finger::AN2KView](#) object represents an ANSI/NIST Type-3/4/5/6 record, and can return the image as well as the other information associated with that image, such as the minutiae from the corresponding Type-9 record.

For these types of records, the image resolution and scan resolution are identical. For compressed images, applications can compare the image resolution and size taken from the Type-3/4/5/6 record to that returned by the [Image](#) object directly.

E.9.2 Constructor & Destructor Documentation

E.9.2.1 [BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution](#) (const std::string filename, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

Construct an AN2K finger view from a file.

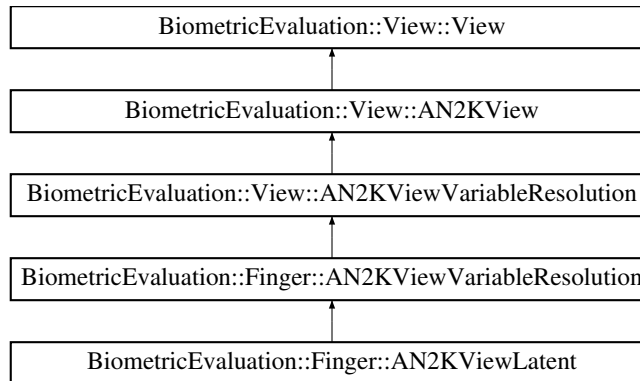
The file must contain the entire AN2K record, not just the finger image and/or minutiae records.

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

- be_finger_an2kview_fixedres.h

E.10 BiometricEvaluation::Finger::AN2KViewLatent Class Reference

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewLatent:



Public Member Functions

- [AN2KViewLatent](#) (const std::string &filename, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)
Construct an AN2K finger view from a file.
- [AN2KViewLatent](#) (Memory::uint8Array &buf, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)
Construct an AN2K finger view using from a memory buffer.
- QualityMetricSet [getLatentQualityMetric](#) () const
Obtain metrics for latent image quality score data for the image stored in this record.
- PositionDescriptors [getSearchPositionDescriptors](#) () const
Return search position descriptors.

E.10.1 Constructor & Destructor Documentation

E.10.1.1 BiometricEvaluation::Finger::AN2KViewLatent::AN2KViewLatent (const std::string & filename, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

Construct an AN2K finger view from a file.

The file must contain the entire AN2K record, not just the finger image and/or minutiae records.

E.10.1.2 BiometricEvaluation::Finger::AN2KViewLatent::AN2KViewLatent (Memory::uint8Array & buf, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

Construct an AN2K finger view using from a memory buffer.

The buffer must contain the entire AN2K record, not just the finger image and/or minutiae records.

E.10.2 Member Function Documentation

E.10.2.1 `QualityMetricSet BiometricEvaluation::Finger::AN2KViewLatent::getLatentQualityMetric () const`

Obtain metrics for latent image quality score data for the image stored in this record.

Returns

Latent quality metrics

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

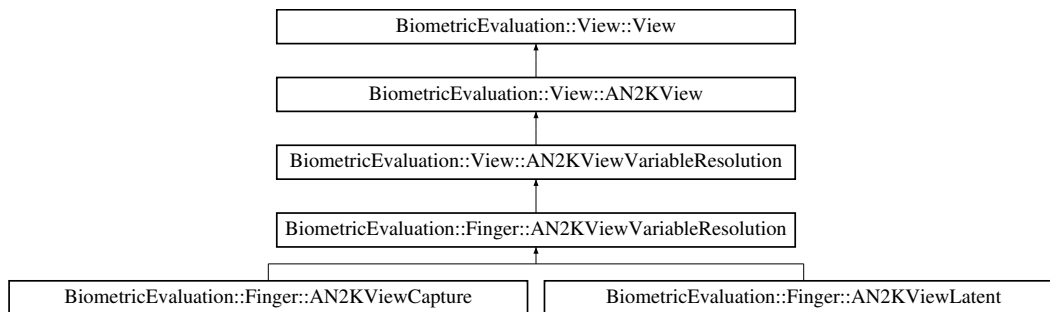
- `be_finger_an2kview_latent.h`

E.11 `BiometricEvaluation::Finger::AN2KViewVariableResolution` Class Reference

A class to represent single finger view based on an ANSI/NIST record.

```
#include <be_finger_an2kview_varres.h>
```

Inheritance diagram for `BiometricEvaluation::Finger::AN2KViewVariableResolution`:



Classes

- struct [PrintPositionCoordinate](#)
Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

Public Types

- typedef struct [PrintPositionCoordinate](#) **PrintPositionCoordinate**
- typedef std::vector < [PrintPositionCoordinate](#) > **PrintPositionCoordinateSet**

Public Member Functions

- `Finger::PositionSet` [getPositions](#) () const
Obtain the set of finger positions.

- Finger::Impression::Kind [getImpressionType](#) () const
Obtain the finger impression code.
- PrintPositionCoordinateSet [getPrintPositionCoordinates](#) () const
Obtain print position coordinates.

Protected Member Functions

- [AN2KViewVariableResolution](#) (const std::string &filename, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)
Construct an AN2K finger view from a file.
- [AN2KViewVariableResolution](#) (Memory::uint8Array &buf, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)
Construct an AN2K finger view using from a memory buffer.
- PositionDescriptors [getPositionDescriptors](#) () const
Obtain the set of position descriptors.

Static Protected Member Functions

- static [PrintPositionCoordinate](#) [convertPrintPositionCoordinate](#) (SUBFIELD *subfield) throw (Error::DataError)
Convert a print position coordinate AN2K subfield to a [PrintPositionCoordinate](#) object.
- static PositionDescriptors [parsePositionDescriptors](#) (const RecordType::Kind typeId, const RECORD *record) throw (Error::DataError)
Parse position descriptors from a record.

E.11.1 Detailed Description

A class to represent single finger view based on an ANSI/NIST record.

The view represents a variable resolution (Type-13, 14) ANSI_NIST record.

E.11.2 Constructor & Destructor Documentation

- E.11.2.1** `BiometricEvaluation::Finger::AN2KViewVariableResolution::AN2KViewVariableResolution (const std::string & filename, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError) [protected]`

Construct an AN2K finger view from a file.

The file must contain the entire AN2K record, not just the finger image and/or minutiae records.

Reimplemented from [BiometricEvaluation::View::AN2KViewVariableResolution](#).

- E.11.2.2** `BiometricEvaluation::Finger::AN2KViewVariableResolution::AN2KViewVariableResolution (Memory::uint8Array & buf, const RecordType::Kind typeId, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError) [protected]`

Construct an AN2K finger view using from a memory buffer.

The buffer must contain the entire AN2K record, not just the finger image and/or minutiae records.

Reimplemented from [BiometricEvaluation::View::AN2KViewVariableResolution](#).

E.11.3 Member Function Documentation

E.11.3.1 `Finger::PositionSet` `BiometricEvaluation::Finger::AN2KViewVariableResolution::getPositions () const`

Obtain the set of finger positions.

An AN2K finger image record contains a set of possible finger positions. This method returns that set as read from the image record. Any minutiae record (Type-9) associated with this image will have its own set of positions.

E.11.3.2 `PrintPositionCoordinateSet` `BiometricEvaluation::Finger::AN2KViewVariableResolution::getPrintPositionCoordinates () const`

Obtain print position coordinates.

Returns

Set of all `PrintPositionCoordinates`

E.11.3.3 `static PrintPositionCoordinate` `BiometricEvaluation::Finger::AN2KViewVariableResolution::convertPrintPositionCoordinate (SUBFIELD * subfield) throw (Error::DataError) [static, protected]`

Convert a print position coordinate AN2K subfield to a [PrintPositionCoordinate](#) object.

Parameters

<code>in</code>	<code><i>subfield</i></code>	A print position coordinate AN2K subfield
-----------------	------------------------------	---

Returns

Object representation of field.

Exceptions

Error::DataError	Invalid data for a print position coordinate AN2K field.
----------------------------------	--

E.11.3.4 `static PositionDescriptors` `BiometricEvaluation::Finger::AN2KViewVariableResolution::parsePositionDescriptors (const RecordType::Kind typeID, const RECORD * record) throw (Error::DataError) [static, protected]`

Parse position descriptors from a record.

Parameters

in	<i>typeID</i>	The logical record type.
in	<i>record</i>	The opened AN2K record.

Returns

Mapping of finger position codes to finger image code.

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

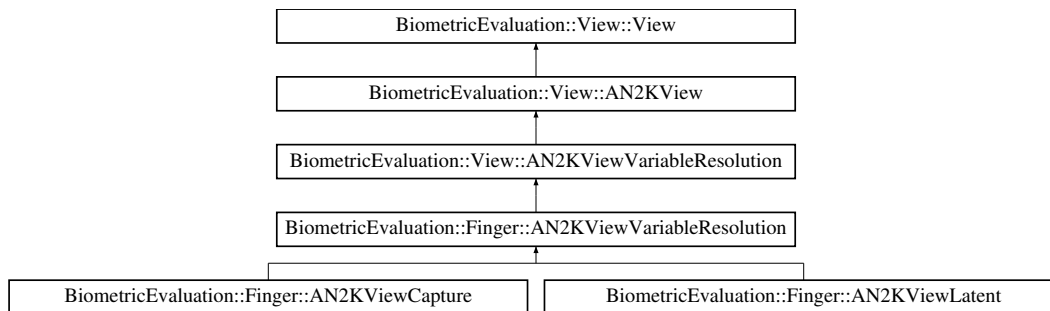
- be_finger_an2kview_varres.h

E.12 BiometricEvaluation::View::AN2KViewVariableResolution Class Reference

A class to represent single view based on an ANSI/NIST record.

```
#include <be_view_an2kview_varres.h>
```

Inheritance diagram for BiometricEvaluation::View::AN2KViewVariableResolution:

**Classes**

- struct [AN2KQualityMetric](#)
A structure to represent an AN2K quality metric.

Public Types

- typedef struct [AN2KQualityMetric](#) **AN2KQualityMetric**
- typedef std::vector < [AN2KQualityMetric](#) > **QualityMetricSet**

Public Member Functions

- string [getSourceAgency](#) () const
Obtain the source agency.
- string [getCaptureDate](#) () const
Obtain the capture date.

- string `getComment ()` const
Obtain the comment field.
- `Memory::uint8Array getUserDefinedField` (const uint16_t field) const throw (Error::ParameterError)
Obtain a user-defined field.

Static Public Member Functions

- static QualityMetricSet `extractQuality` (FIELD *field) throw (Error::DataError)
Read a Quality Metric Set from a variable resolution AN2K record.
- static `Memory::uint8Array parseUserDefinedField` (const RECORD *const record, int fieldID) throw (Error::ParameterError)
Read raw bytes from a user-defined AN2K field.

Protected Member Functions

- `AN2KViewVariableResolution` (const std::string &filename, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)
Construct an AN2K finger view from a file.
- `AN2KViewVariableResolution` (`Memory::uint8Array` &buf, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)
Construct an AN2K finger view using from a memory buffer.
- QualityMetricSet `getQualityMetric ()` const
Obtain quality metrics for associated image record.

E.12.1 Detailed Description

A class to represent single view based on an ANSI/NIST record.

The view represents a variable resolution (Type-13/14/15) AN2K record.

E.12.2 Constructor & Destructor Documentation

E.12.2.1 `BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution (const std::string & filename, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError) [protected]`

Construct an AN2K finger view from a file.

The file must contain the entire AN2K record, not just the finger image and/or minutiae records.

Reimplemented in `BiometricEvaluation::Finger::AN2KViewVariableResolution`.

E.12.2.2 `BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution (Memory::uint8Array & buf, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError) [protected]`

Construct an AN2K finger view using from a memory buffer.

The buffer must contain the entire AN2K record, not just the finger image and/or minutiae records.

Reimplemented in [BiometricEvaluation::Finger::AN2KViewVariableResolution](#).

E.12.3 Member Function Documentation

E.12.3.1 `static QualityMetricSet BiometricEvaluation::View::AN2KViewVariableResolution::extractQuality (FIELD * field) throw (Error::DataError) [static]`

Read a Quality Metric Set from a variable resolution AN2K record.

Parameters

<code>in</code>	<code><i>field</i></code>	A pointer to the field within the AN2K record.
-----------------	---------------------------	--

Exceptions

<code>Error::DataError</code>	The data contains an invalid value.
---	-------------------------------------

E.12.3.2 `string BiometricEvaluation::View::AN2KViewVariableResolution::getComment () const`

Obtain the comment field.

The comment field is optional in an AN2K record.

Returns

The comment field, empty string if not present.

E.12.3.3 `Memory::uint8Array BiometricEvaluation::View::AN2KViewVariableResolution::getUserDefined-Field (const uint16_t field) const throw (Error::ParameterError)`

Obtain a user-defined field.

Fields are retrieved on-demand and then cached.

Parameters

<code>in</code>	<code><i>field</i></code>	The field number to retrieve.
-----------------	---------------------------	-------------------------------

Returns

Raw bytes read from the field.

Exceptions

<code>Error::ParameterError</code>	Invalid value for field.
--	--------------------------

E.12.3.4 `static Memory::uint8Array BiometricEvaluation::View::AN2KViewVariableResolution::parseUserDefinedField (const RECORD *const record, int fieldID) throw (Error::ParameterError)`
`[static]`

Read raw bytes from a user-defined AN2K field.

Parameters

in	<i>record</i>	Pointer to a RECORD containing the user-defined field.
in	<i>fieldID</i>	The user-defined field number.

Returns

Raw bytes from field.

Exceptions

<i>Error::ParameterError</i>	Invalid value for fieldID.
--	----------------------------

E.12.3.5 `QualityMetricSet BiometricEvaluation::View::AN2KViewVariableResolution::getQualityMetric ()`
`const [protected]`

Obtain quality metrics for associated image record.

Returns

Quality metrics

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

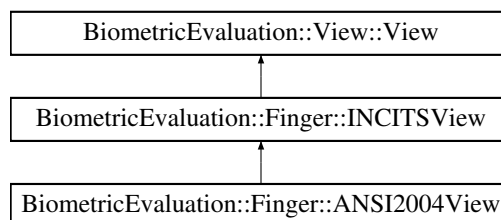
- `be_view_an2kview_varres.h`

E.13 BiometricEvaluation::Finger::ANSI2004View Class Reference

A class to represent single finger view and derived information.

```
#include <be_finger_ansi2004view.h>
```

Inheritance diagram for `BiometricEvaluation::Finger::ANSI2004View`:



Public Member Functions

- [ANSI2004View](#) ()
Construct an empty ANSI finger view.
- [ANSI2004View](#) (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view-Number) throw (Error::DataError, Error::FileError)
Construct an ANSI-2004 finger view from records contained in files.
- [ANSI2004View](#) (Memory::uint8Array &fmrBuffer, Memory::uint8Array &firBuffer, const uint32_t view-Number) throw (Error::DataError)
Construct an ANSI-2004 finger view from records contained in buffers.

Static Public Attributes

- static const uint16_t **CORE_TYPE_MASK** = 0xC0
- static const uint16_t **CORE_TYPE_SHIFT** = 6
- static const uint16_t **CORE_NUM_CORES_MASK** = 0x0F
- static const uint16_t **CORE_X_COORD_MASK** = 0x3FFF
- static const uint16_t **CORE_Y_COORD_MASK** = 0x3FFF
- static const uint16_t **DELTA_TYPE_MASK** = 0xC0
- static const uint16_t **DELTA_TYPE_SHIFT** = 6
- static const uint16_t **DELTA_NUM_DELTAS_MASK** = 0x3F
- static const uint16_t **DELTA_X_COORD_MASK** = 0x3FFF
- static const uint16_t **DELTA_Y_COORD_MASK** = 0x3FFF

Protected Member Functions

- virtual void [readCoreDeltaData](#) (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::Core-PointSet &cores, Feature::DeltaPointSet &deltas) throw (Error::DataError)
Read the core points data.

E.13.1 Detailed Description

A class to represent single finger view and derived information.

A [Finger::ANSI2004View](#) object represents a finger view from a INCITS/ANSI-2004 [Finger](#) Minutiae Record.

E.13.2 Constructor & Destructor Documentation

- E.13.2.1** [BiometricEvaluation::Finger::ANSI2004View::ANSI2004View](#) (const std::string & fmrFilename, const std::string & firFilename, const uint32_t viewNumber) throw (Error::DataError, Error::FileError)

Construct an ANSI-2004 finger view from records contained in files.

A view can be constructed from a single record, with information missing as appropriate. For example, if a view is constructed with just the minutiae record, no image would be part of the view. However, the image size etc. would be present because that information is also present in the minutiae record.

Parameters

in	<i>fmrFilename</i>	The name of the file containing the complete finger minutiae record.
in	<i>firFilename</i>	The name of the file containing the complete finger image record.
in	<i>viewNumber</i>	The finger view number to use.

E.13.2.2 BiometricEvaluation::Finger::ANSI2004View::ANSI2004View (Memory::uint8Array & *fmrBuffer*, Memory::uint8Array & *firBuffer*, const uint32_t *viewNumber*) throw (Error::DataError)

Construct an ANSI-2004 finger view from records contained in buffers.

A view can be constructed from a single record, with information missing as appropriate. For example, if a view is constructed with just the minutiae record, no image would be part of the view. However, the image size etc. would be present because that information is also present in the minutiae record.

Parameters

in	<i>fmrBuffer</i>	The buffer containing the complete finger minutiae record.
in	<i>firBuffer</i>	The buffer containing the complete finger image record.
in	<i>viewNumber</i>	The finger view number to use.

E.13.3 Member Function Documentation

E.13.3.1 virtual void BiometricEvaluation::Finger::ANSI2004View::readCoreDeltaData (Memory::IndexedBuffer & *buf*, uint32_t *dataLength*, Feature::CorePointSet & *cores*, Feature::DeltaPointSet & *deltas*) throw (Error::DataError) [protected, virtual]

Read the core points data.

This method must be overridden by derived classes to read data in a specific record format.

Parameters

in, out	<i>buf</i>	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last core point data item.
out	<i>cores</i>	The set of core data items.
out	<i>deltas</i>	The set of delta data items.
in	<i>dataLength</i>	The length of the entire ridge count data block.

Implements [BiometricEvaluation::Finger::INCITSView](#).

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

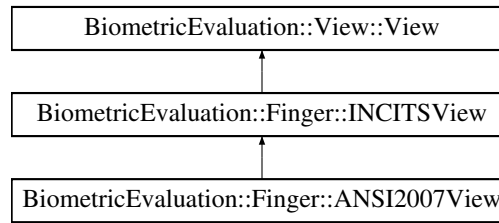
- `be_finger_ansi2004view.h`

E.14 BiometricEvaluation::Finger::ANSI2007View Class Reference

A class to represent single finger view and derived information.

```
#include <be_finger_ansi2007view.h>
```

Inheritance diagram for BiometricEvaluation::Finger::ANSI2007View:



Public Member Functions

- [ANSI2007View](#) (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view-Number) throw (Error::DataError, Error::FileError)
Construct an ANSI-2007 finger view from records contained in files.
- [ANSI2007View](#) (Memory::uint8Array &fmrBuffer, Memory::uint8Array &firBuffer, const uint32_t view-Number) throw (Error::DataError)
Construct an ANSI-2007 finger view from records contained in buffers.

Static Public Attributes

- static const string **FMR_SPEC_VERSION**
- static const uint16_t **CORE_TYPE_MASK** = 0xC0
- static const uint16_t **CORE_TYPE_SHIFT** = 6
- static const uint16_t **CORE_NUM_CORES_MASK** = 0x0F
- static const uint16_t **CORE_X_COORD_MASK** = 0x3FFF
- static const uint16_t **CORE_Y_COORD_MASK** = 0x3FFF
- static const uint16_t **DELTA_TYPE_MASK** = 0xC0
- static const uint16_t **DELTA_TYPE_SHIFT** = 6
- static const uint16_t **DELTA_NUM_DELTAS_MASK** = 0x0F
- static const uint16_t **DELTA_X_COORD_MASK** = 0x3FFF
- static const uint16_t **DELTA_Y_COORD_MASK** = 0x3FFF

Protected Member Functions

- void [readFMRHeader](#) (Memory::IndexedBuffer &buf, const uint32_t formatStandard) throw (Error::ParameterError, Error::DataError)
Read the common finger minutiae record header from an INCITS record.
- void [readFVMR](#) (Memory::IndexedBuffer &buf) throw (Error::DataError)
Read the common finger view record information from an INCITS record.
- virtual void [readCoreDeltaData](#) (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas) throw (Error::DataError)
Read the core points data.

E.14.1 Detailed Description

A class to represent single finger view and derived information.

A [Finger::ANSI2007View](#) object represents a finger view from a INCITS/ANSI-2007 [Finger](#) Minutiae Record.

E.14.2 Constructor & Destructor Documentation

E.14.2.1 BiometricEvaluation::Finger::ANSI2007View::ANSI2007View (const std::string & *fmrFilename*, const std::string & *firFilename*, const uint32_t *viewNumber*) throw (Error::DataError, Error::FileError)

Construct an ANSI-2007 finger view from records contained in files.

A view can be constructed from a single record, with information missing as appropriate. For example, if a view is constructed with just the minutiae record, no image would be part of the view. However, the image size etc. would be present because that information is also present in the minutiae record.

Parameters

in	<i>fmrFilename</i>	The name of the file containing the complete finger minutiae record.
in	<i>firFilename</i>	The name of the file containing the complete finger image record.
in	<i>viewNumber</i>	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.
----------------------------------	------------------------

E.14.2.2 BiometricEvaluation::Finger::ANSI2007View::ANSI2007View (Memory::uint8Array & *fmrBuffer*, Memory::uint8Array & *firBuffer*, const uint32_t *viewNumber*) throw (Error::DataError)

Construct an ANSI-2007 finger view from records contained in buffers.

A view can be constructed from a single record, with information missing as appropriate. For example, if a view is constructed with just the minutiae record, no image would be part of the view. However, the image size etc. would be present because that information is also present in the minutiae record.

Parameters

in	<i>fmrBuffer</i>	The buffer containing the complete finger minutiae record.
in	<i>firBuffer</i>	The buffer containing the complete finger image record.
in	<i>viewNumber</i>	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.
----------------------------------	------------------------

E.14.3 Member Function Documentation

E.14.3.1 `void BiometricEvaluation::Finger::ANSI2007View::readFMRHeader (Memory::IndexedBuffer & buf, const uint32_t formatStandard) throw (Error::ParameterError, Error::DataError) [protected]`

Read the common finger minutiae record header from an INCITS record.

For ANSI-2004 and ISO-2005 record formats, the finger minutiae record header is (mostly) the same.

Parameters

in	<i>buf</i>	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the header.
in	<i>formatStandard</i>	Value indicating which header version to read; one of FMR_ANSI2004_STANDARD or FMR_ISO2005_STANDARD.

Exceptions

<i>ParameterError</i>	The specVersion parameter is incorrect.
<i>DataError</i>	The INCITS record has invalid or missing data.

Reimplemented from [BiometricEvaluation::Finger::INCITSView](#).

E.14.3.2 `void BiometricEvaluation::Finger::ANSI2007View::readFVMR (Memory::IndexedBuffer & buf) throw (Error::DataError) [protected]`

Read the common finger view record information from an INCITS record.

A [Finger View](#) from an INCITS record includes image information, minutiae, and extended data ridge counts, cores/deltas, etc.) For ANSI-2004 and ISO-2005 record formats, the finger view representation is the same, so this functions parses those record formats. The minutiae data items are also read, as well as any extended data.

Parameters

in, out	<i>buf</i>	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the finger view, including the extended data.
---------	------------	--

Exceptions

<i>DataError</i>	The INCITS record has invalid or missing data.
------------------	--

Reimplemented from [BiometricEvaluation::Finger::INCITSView](#).

E.14.3.3 `virtual void BiometricEvaluation::Finger::ANSI2007View::readCoreDeltaData (Memory::IndexedBuffer & buf, uint32_t dataLength, Feature::CorePointSet & cores, Feature::DeltaPointSet & deltas) throw (Error::DataError) [protected, virtual]`

Read the core points data.

This method must be overridden by derived classes to read data in a specific record format.

Parameters

in, out	<i>buf</i>	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last core point data item.
out	<i>cores</i>	The set of core data items.
out	<i>deltas</i>	The set of delta data items.
in	<i>dataLength</i>	The length of the entire ridge count data block.

Implements [BiometricEvaluation::Finger::INCITSView](#).

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

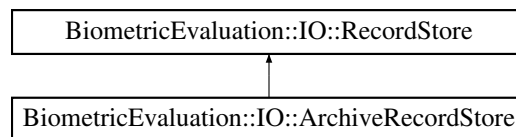
- `be_finger_ansi2007view.h`

E.15 BiometricEvaluation::IO::ArchiveRecordStore Class Reference

This class implements the [IO::RecordStore](#) interface by storing data items in single file, with an associated manifest file.

```
#include <be_io_archiverecstore.h>
```

Inheritance diagram for BiometricEvaluation::IO::ArchiveRecordStore:



Public Member Functions

- [ArchiveRecordStore](#) (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- [ArchiveRecordStore](#) (const string &name, const string &parentDir, uint8_t mode=IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- [~ArchiveRecordStore](#) ()
- uint64_t [getSpaceUsed](#) () const throw (Error::StrategyError)
Obtain real storage utilization.
- void [sync](#) () const throw (Error::StrategyError)
- 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)
- void [replace](#) (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t [length](#) (const string &key) const throw (Error::ObjectDoesNotExist)
- void [flush](#) (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t [sequence](#) (string &key, void *const data=NULL, int cursor=[BE_RECSTORE_SEQ_NEXT](#)) throw (Error::ObjectDoesNotExist, Error::StrategyError)

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

- void [setCursorAtKey](#) (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void [changeName](#) (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- bool [needsVacuum](#) ()
- string [getArchiveName](#) () const
- string [getManifestName](#) () const

Static Public Member Functions

- static bool [needsVacuum](#) (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- static void [vacuum](#) (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)

E.15.1 Detailed Description

This class implements the [IO::RecordStore](#) interface by storing data items in single file, with an associated manifest file.

Archives consist of binary records written back to back of each other. To pull information out of an archive, a manifest file is written in the same directory as the archive file.

Each record is assigned a string key, which will be required for retrieving the data. As the data is written, a plain text entry is entered into the manifest in the format:

key offset size

where offset is the offset into the archive file key's data chunk resides and size is the length of key's data chunk.

By default, information is not removed when updated in the archive, rather the old information is ignored. Therefore, it is possible to have multiple entries in the manifest for one key. The last entry for the key is considered accurate. If the last offset for a key is ARCHIVE_RECORD_REMOVED, the information is treated as unavailable.

E.15.2 Constructor & Destructor Documentation

E.15.2.1 BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore (const string & name, const string & description, const string & parentDir) throw (Error::ObjectExists, Error::StrategyError)

Create a new [ArchiveRecordStore](#), 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

Error::ObjectExists	The store already exists.
Error::StrategyError	An error occurred when accessing the underlying file system.

E.15.2.2 `BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore (const string & name, const string & parentDir, uint8_t mode = IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)`

Open an existing [ArchiveRecordStore](#).

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

Error::ObjectDoesNotExist	The store does not exist.
Error::StrategyError	An error occurred when accessing the underlying file system.

E.15.2.3 `BiometricEvaluation::IO::ArchiveRecordStore::~~ArchiveRecordStore ()`

Destructor.

E.15.3 Member Function Documentation

E.15.3.1 `uint64_t BiometricEvaluation::IO::ArchiveRecordStore::getSpaceUsed () const throw (Error::StrategyError) [virtual]`

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the [RecordStore](#).

Exceptions

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

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.15.3.2 `void BiometricEvaluation::IO::ArchiveRecordStore::sync () const throw (Error::StrategyError) [virtual]`

Synchronize the entire record store to persistent storage.

Exceptions

<i>Error::StrategyError</i>	An error occurred when using the underlying storage system.
---	---

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.15.3.3 void BiometricEvaluation::IO::ArchiveRecordStore::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

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.15.3.4 void BiometricEvaluation::IO::ArchiveRecordStore::remove (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Remove a record from the store.

Parameters

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

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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.15.3.5 uint64_t BiometricEvaluation::IO::ArchiveRecordStore::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	<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

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.15.3.6 `void BiometricEvaluation::IO::ArchiveRecordStore::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	<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.15.3.7 `uint64_t BiometricEvaluation::IO::ArchiveRecordStore::length (const string & key) const throw (Error::ObjectDoesNotExist) [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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.15.3.8 `void BiometricEvaluation::IO::ArchiveRecordStore::flush (const string & key) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.15.3.9 `uint64_t BiometricEvaluation::IO::ArchiveRecordStore::sequence (string & key, void *const data = NULL, int cursor = BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Sequence through a [RecordStore](#), returning the key/data pairs.

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

Parameters

out	<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

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.15.3.10 void BiometricEvaluation::IO::ArchiveRecordStore::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 sequence() .
----	------------	---

Exceptions

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.15.3.11 void BiometricEvaluation::IO::ArchiveRecordStore::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 RecordStore .
----	-------------	--

Exceptions

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

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.15.3.12 bool BiometricEvaluation::IO::ArchiveRecordStore::needsVacuum ()

See if the [ArchiveRecordStore](#) would benefit from calling [vacuum\(\)](#) to remove deleted entries, since [vacuum\(\)](#) is an expensive operation.

Returns

true if `vacuum()` would be beneficial false otherwise

E.15.3.13 `static bool BiometricEvaluation::IO::ArchiveRecordStore::needsVacuum (const string & name, const string & parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]`

See if the `ArchiveRecordStore` would benefit from calling `vacuum()` to remove deleted entries, since `vacuum()` is an expensive operation.

Parameters

in	<i>name</i>	The name of the existing <code>RecordStore</code> .
in	<i>parentDir</i>	Where, in the filesystem, the store is rooted.

Exceptions

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

Returns

true if `vacuum()` would be beneficial false otherwise

E.15.3.14 `static void BiometricEvaluation::IO::ArchiveRecordStore::vacuum (const string & name, const string & parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]`

Remove deleted entries from the manifest and archive files to save space on disk.

Parameters

in	<i>name</i>	The name of the existing <code>RecordStore</code> .
in	<i>parentDir</i>	Where, in the file system, the store is rooted.

Exceptions

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

Note

This is an expensive operation.

E.15.3.15 `string BiometricEvaluation::IO::ArchiveRecordStore::getArchiveName () const`

Obtain the name of the file storing the data for this store.

Returns

Path to archive file.

E.15.3.16 `string BiometricEvaluation::IO::ArchiveRecordStore::getManifestName () const`

Obtain the name of the file storing the manifest data data for this store.

Returns

Path to manifest file.

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

- `be_io_archiverecstore.h`

E.16 `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::MemoryError)

Change the number of accessible elements.
- void [copy](#) (const_iterator buffer)

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

Deep-copy the contents of a buffer into this [AutoArray](#).
- [AutoArray](#) (size_type size=0) throw (Error::MemoryError)

Construct an [AutoArray](#).
- [AutoArray](#) (const [AutoArray](#) ©) throw (Error::MemoryError)

Construct an [AutoArray](#).
- [AutoArray](#) & operator= (const [AutoArray](#) &other) throw (Error::MemoryError)

Assignment operator overload performing a deep copy.
- [~AutoArray](#) ()

E.16.1 Detailed Description

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

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

E.16.2 Member Typedef Documentation

E.16.2.1 `template<class T> typedef T BiometricEvaluation::Memory::AutoArray< T >::value_type`

Type of element

E.16.2.2 `template<class T> typedef size_t BiometricEvaluation::Memory::AutoArray< T >::size_type`

Type of subscripts, counts, etc.

E.16.2.3 `template<class T> typedef T* BiometricEvaluation::Memory::AutoArray< T >::iterator`

Iterator of element

E.16.2.4 `template<class T> typedef const T* BiometricEvaluation::Memory::AutoArray< T >::const_iterator`

Const iterator of element

E.16.2.5 `template<class T> typedef T& BiometricEvaluation::Memory::AutoArray< T >::reference`

Reference to element

E.16.2.6 `template<class T> typedef const T& BiometricEvaluation::Memory::AutoArray< T >::const_reference`

Const reference element

E.16.3 Constructor & Destructor Documentation

E.16.3.1 `template<class T > BiometricEvaluation::Memory::AutoArray< T >::AutoArray (size_type size = 0) throw (Error::MemoryError)`

Construct an [AutoArray](#).

Parameters

<i>in</i>	<i>size</i>	The number of elements this AutoArray should initially hold.
-----------	-------------	--

Exceptions

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

E.16.3.2 `template<class T > BiometricEvaluation::Memory::AutoArray< T >::AutoArray (const AutoArray< T > & copy) throw (Error::MemoryError)`

Construct an [AutoArray](#).

Parameters

<i>in</i>	<i>copy</i>	An AutoArray whose contents will be deep copied into the new AutoArray .
-----------	-------------	--

Exceptions

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

E.16.3.3 `template<class T > BiometricEvaluation::Memory::AutoArray< T >::~~AutoArray ()`

Destructor

E.16.4 Member Function Documentation**E.16.4.1** `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.16.4.2 `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.16.4.3 `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.16.4.4 `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.16.4.5 `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

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

Returns

Reference to the element at the specified index.

Exceptions

<i>out_of_range</i>	Specified index is outside the bounds of this AutoArray .
---------------------	---

E.16.4.6 `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 AutoArray .
---------------------	---

E.16.4.7 `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.16.4.8 `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.16.4.9 `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.16.4.10 `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.16.4.11 `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.16.4.12 `template<class T > void BiometricEvaluation::Memory::AutoArray< T >::resize (size_type
new_size, bool free = false) throw (Error::MemoryError)`

Change the number of accessible elements.

Parameters

in	<i>new_size</i>	The number of elements the AutoArray 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

Error::MemoryError	Problem allocating memory.
------------------------------------	----------------------------

E.16.4.13 `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 size() bytes will be copied.
----	---------------	--

Warning

If buffer is smaller in size than the current size of the [AutoArray](#), you MUST call [copy\(const_iterator, size_type\)](#). This method must only be used when buffer is larger than or equal to the size of the [AutoArray](#).

E.16.4.14 `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.

Warning

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

E.16.4.15 `template<class T > BiometricEvaluation::Memory::AutoArray< T > & BiometricEvaluation::Memory::AutoArray< T >::operator= (const AutoArray< T > & other) throw (Error::MemoryError)`

Assignment operator overload performing a deep copy.

Parameters

in	<i>other</i>	AutoArray to be copied.
----	--------------	---

Returns

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

Exceptions

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

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

- be_memory_autoarray.h

E.17 BiometricEvaluation::Memory::AutoBuffer< T > Class Template Reference

Public Types

- typedef T [value_type](#)
Manage a memory buffer.
- typedef T & **reference**
- typedef const T & **const_reference**

Public Member Functions

- **operator T * ()**
- T * **operator-> ()**
- [AutoBuffer](#) & **operator=** (const [AutoBuffer](#) &other)
- [AutoBuffer](#) (T *data)
- [AutoBuffer](#) (int(*ctor)(T **), void(*dtor)(T *), int(*copyCtor)(T **, T *)=NULL)
- [AutoBuffer](#) (const [AutoBuffer](#) ©)

```
template<class T> class BiometricEvaluation::Memory::AutoBuffer< T >
```

E.17.1 Member Typedef Documentation

E.17.1.1 template<class T> typedef T BiometricEvaluation::Memory::AutoBuffer< T >::value_type

Manage a memory buffer.

It's easier to think of [AutoBuffer](#) as a wrapper for a pointer rather than the object it truly is. Therefore, you can interact with the [AutoBuffer](#) object exactly how you would a traditional pointer, without worrying about memory management.

Say you wanted to use an ANSI_NIST* but didn't want to be responsible for allocating or freeing the memory. Create an [AutoBuffer](#) object like:

```
AutoBuffer<ANSI_NIST> obj = AutoBuffer(allocator_fn, deallocator_fn[, copy_constructor]);
```

Notice the [AutoBuffer](#) is for ANSI_NIST and not ANSI_NIST*, since [AutoBuffer](#) will handle the pointer for you. You can pass the [AutoBuffer](#)<ANSI_NIST> object to any function that takes an ANSI_NIST*. For example, it's perfectly valid to pass our 'obj' object above to:

```
write_fmttext(FILE *, ANSI_NIST *)
```

If you want to access a member from 'obj', you can use the dereference operator just like you would on a regular ANSI_NIST*:

```
int size = obj->num_bytes;
```

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

- be_memory_autobuffer.h

E.18 BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet Struct Reference

Public Member Functions

- [CharacterSet](#) (uint16_t [identifier](#)=0, string [commonName](#)="", string [version](#)="")
Create a new [CharacterSet](#) struct.

Public Attributes

- uint16_t [identifier](#)
- string [commonName](#)
- string [version](#)

E.18.1 Constructor & Destructor Documentation

E.18.1.1 [BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::CharacterSet](#) (uint16_t *identifier* = 0, string *commonName* = "", string *version* = "") `[inline]`

Create a new [CharacterSet](#) struct.

Parameters

<i>identifier</i>	Numeric identifier of the character set.
<i>commonName</i>	Common name of the character set.
<i>version</i>	Optional version number of the character set.

E.18.2 Member Data Documentation

E.18.2.1 `uint16_t` [BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::identifier](#)

Identifier (000-999)

E.18.2.2 `string` [BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::commonName](#)

Common name of the character set

E.18.2.3 string BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::version

Optional version of the character set

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

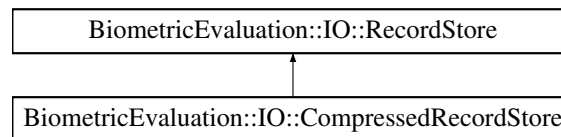
- be_data_interchange_an2k.h

E.19 BiometricEvaluation::IO::CompressedRecordStore Class Reference

Sibling-implemented [RecordStore](#) with Compression.

```
#include <be_io_compressedrecstore.h>
```

Inheritance diagram for BiometricEvaluation::IO::CompressedRecordStore:



Public Member Functions

- [CompressedRecordStore](#) (const string &name, const string &description, const string &recordStoreType, const string &parentDir, const string &compressorType) throw (Error::ObjectExists, Error::StrategyError)
- [CompressedRecordStore](#) (const string &name, const string &description, const string &recordStoreType, const string &parentDir, const [Compressor::Kind](#) &compressorType) throw (Error::ObjectExists, Error::StrategyError)
- [CompressedRecordStore](#) (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 [sync](#) () const throw (Error::StrategyError)
- 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)
- void [replace](#) (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t [length](#) (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void [flush](#) (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t [sequence](#) (string &key, void *const data=NULL, int cursor=[BE_RECSTORE_SEQ_NEXT](#)) throw (Error::ObjectDoesNotExist, Error::StrategyError)
Sequence through a [RecordStore](#), returning the key/data pairs.
- void [setCursorAtKey](#) (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void [changeName](#) (const string &name) throw (Error::ObjectExists, Error::StrategyError)

Static Public Attributes

- static const string [BACKING_STORE](#)
- static const string [COMPRESSOR_TYPE_KEY](#)

E.19.1 Detailed Description

Sibling-implemented [RecordStore](#) with Compression.

E.19.2 Constructor & Destructor Documentation

E.19.2.1 `BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const string & name, const string & description, const string & recordStoreType, const string & parentDir, const string & compressorType) throw (Error::ObjectExists, Error::StrategyError)`

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

Parameters

in	<i>name</i>	The name of the store.
in	<i>description</i>	The store's description.
in	<i>recordStoreType</i>	The type of RecordStore subclass the internal RecordStores should be.
in	<i>parentDir</i>	The directory where the store is to be created.
in	<i>compressorType</i>	The type of compression that should be used within the internal RecordStores.

Exceptions

Error::ObjectExists	The store already exists.
Error::StrategyError	An error occurred when accessing the underlying file system.

E.19.2.2 `BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const string & name, const string & description, const string & recordStoreType, const string & parentDir, const Compressor::Kind & compressorType) throw (Error::ObjectExists, Error::StrategyError)`

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

Parameters

in	<i>name</i>	The name of the store.
in	<i>description</i>	The store's description.
in	<i>recordStoreType</i>	The type of RecordStore subclass the internal RecordStores should be.
in	<i>parentDir</i>	The directory where the store is to be created.
in	<i>compressorType</i>	The type of compression that should be used within the internal RecordStores.

Exceptions

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

E.19.2.3 `BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const string & name, const string & parentDir, uint8_t mode = IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)`

Open an existing [CompressedRecordStore](#).

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

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

E.19.3 Member Function Documentation

E.19.3.1 `uint64_t BiometricEvaluation::IO::CompressedRecordStore::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

<i>Error::StrategyError</i>	An error occurred when using the underlying storage system.
---	---

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.19.3.2 `void BiometricEvaluation::IO::CompressedRecordStore::sync () const throw (Error::StrategyError) [virtual]`

Synchronize the entire record store to persistent storage.

Exceptions

<i>Error::StrategyError</i>	An error occurred when using the underlying storage system.
---	---

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.19.3.3 `void BiometricEvaluation::IO::CompressedRecordStore::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

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.19.3.4 `void BiometricEvaluation::IO::CompressedRecordStore::remove (const string & key) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Remove a record from the store.

Parameters

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

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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.19.3.5 `uint64_t BiometricEvaluation::IO::CompressedRecordStore::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	<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

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.19.3.6 `void BiometricEvaluation::IO::CompressedRecordStore::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	<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.19.3.7 `uint64_t BiometricEvaluation::IO::CompressedRecordStore::length (const string & key) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.19.3.8 `void BiometricEvaluation::IO::CompressedRecordStore::flush (const string & key) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.19.3.9 `uint64_t BiometricEvaluation::IO::CompressedRecordStore::sequence (string & key, void *const data = NULL, int cursor = BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Sequence through a [RecordStore](#), returning the key/data pairs.

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

Parameters

out	<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

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.19.3.10 void BiometricEvaluation::IO::CompressedRecordStore::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 sequence() .
----	------------	---

Exceptions

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.19.3.11 void BiometricEvaluation::IO::CompressedRecordStore::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 RecordStore .
----	-------------	--

Exceptions

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

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.19.4 Member Data Documentation

E.19.4.1 const string BiometricEvaluation::IO::CompressedRecordStore::BACKING_STORE [static]

Name of the underlying store within this RS

E.19.4.2 const string BiometricEvaluation::IO::CompressedRecordStore::COMPRESSOR_TYPE_KEY [static]

Name of the key storing compressor type

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

- `be_io_compressedrecstore.h`

E.20 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.20.1 Detailed Description

[Image](#) compression algorithms.

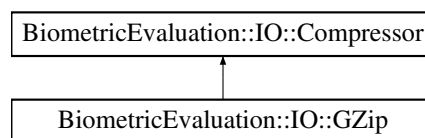
The documentation for this class was generated from the following file:

- `be_image.h`

E.21 BiometricEvaluation::IO::Compressor Class Reference

```
#include <be_io_compressor.h>
```

Inheritance diagram for BiometricEvaluation::IO::Compressor:



Public Types

- enum **Kind** { **GZIP** }

Public Member Functions

- [Compressor](#) ()
Create a new *Compressor* object.
- virtual [Memory::uint8Array compress](#) (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize) const =0 throw (Error::StrategyError)
Compress a buffer.
- virtual [Memory::uint8Array compress](#) (const [Memory::uint8Array](#) &uncompressedData) const =0 throw (Error::StrategyError)

Compress a buffer.

- virtual void [compress](#) (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize, const string &outputFile) const =0 throw (Error::ObjectExists, Error::StrategyError)

Compress a buffer.

- virtual void [compress](#) (const [Memory::uint8Array](#) &uncompressedData, const string &outputFile) const =0 throw (Error::ObjectExists, Error::StrategyError)

Compress a buffer.

- virtual [Memory::uint8Array](#) [compress](#) (const string &inputFile) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

Compress a file.

- virtual void [compress](#) (const string &inputFile, const string &outputFile) const =0 throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError)

Compress a file.

- virtual [Memory::uint8Array](#) [decompress](#) (const uint8_t *const compressedData, uint64_t compressedDataSize) const =0 throw (Error::StrategyError)

Decompress a compressed buffer.

- virtual [Memory::uint8Array](#) [decompress](#) (const [Memory::uint8Array](#) &compressedData) const =0 throw (Error::StrategyError)

Decompress a compressed buffer.

- virtual [Memory::uint8Array](#) [decompress](#) (const string &inputFile) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

Decompress a compressed buffer into a file.

- virtual void [decompress](#) (const [Memory::uint8Array](#) &compressedData, const string &outputFile) const =0 throw (Error::ObjectExists, Error::StrategyError)

Decompress a file.

- virtual void [decompress](#) (const uint8_t *const compressedData, const uint64_t compressedDataSize, const string &outputFile) const =0 throw (Error::ObjectExists, Error::StrategyError)

Decompress a file.

- virtual void [decompress](#) (const string &inputFile, const string &outputFile) const =0 throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError)

Decompress a file.

- void [setOption](#) (const string &optionName, const string &optionValue) throw (Error::StrategyError)

Assign a compressor option.

- void [setOption](#) (const string &optionName, int64_t optionValue) throw (Error::StrategyError)

Assign a compressor option.

- string [getOption](#) (const string &optionName) const throw (Error::ObjectDoesNotExist)

Obtain a compressor option as an integer.

- int64_t [getOptionAsInteger](#) (const string &optionName) const throw (Error::ObjectDoesNotExist)

Obtain a compressor option as an integer.

- void [removeOption](#) (const string &optionName) throw (Error::ObjectDoesNotExist)

Remove a compressor option.

- virtual [~Compressor](#) ()

Static Public Member Functions

- static string [kindToString](#) ([Compressor::Kind](#) compressor) throw (Error::ObjectDoesNotExist)
Convert Kind enumeration to string.
- static [Compressor::Kind stringToKind](#) (const string &compressor) throw (Error::ObjectDoesNotExist)
Convert string to Kind enumeration.
- static tr1::shared_ptr < [Compressor](#) > [createCompressor](#) ([Compressor::Kind](#) compressorKind=[Compressor::GZIP](#)) throw (Error::ObjectDoesNotExist)

Static Public Attributes

- static const string [GZIPTYPE](#)

E.21.1 Detailed Description

Implementations for compressing and decompressing data

E.21.2 Member Enumeration Documentation

E.21.2.1 enum [BiometricEvaluation::IO::Compressor::Kind](#)

Kinds of Compressors (for factory)

E.21.3 Constructor & Destructor Documentation

E.21.3.1 [BiometricEvaluation::IO::Compressor::Compressor](#) ()

Create a new [Compressor](#) object.

Default compression options will be used.

E.21.3.2 virtual [BiometricEvaluation::IO::Compressor::~Compressor](#) () [virtual]

Destructor

E.21.4 Member Function Documentation

E.21.4.1 static string [BiometricEvaluation::IO::Compressor::kindToString](#) ([Compressor::Kind](#) *compressor*) throw (Error::ObjectDoesNotExist) [static]

Convert Kind enumeration to string.

Parameters

in	<i>compressor</i>	The Compressor to convert.
----	-------------------	--

Returns

String representation of compressor.

Exceptions

<i>Error::ObjectDoesNotExist</i>	compressor is not a valid Compressor type.
----------------------------------	--

E.21.4.2 static [Compressor::Kind](#) [BiometricEvaluation::IO::Compressor::stringToKind](#) (const string & *compressor*) throw (Error::ObjectDoesNotExist) [static]

Convert string to Kind enumeration.

Parameters

in	<i>compressor</i>	The Compressor to convert.
----	-------------------	--

Returns

Kind enumeration of compressor.

Exceptions

<i>Error::ObjectDoesNotExist</i>	compressor is not a valid Compressor type.
----------------------------------	--

E.21.4.3 virtual [Memory::uint8Array](#) [BiometricEvaluation::IO::Compressor::compress](#) (const [uint8_t](#) *const *uncompressedData*, [uint64_t](#) *uncompressedDataSize*) const throw (Error::StrategyError) [pure virtual]

Compress a buffer.

Parameters

<i>uncompressed-Data</i>	Uncompressed data buffer to compress.
<i>uncompressed-DataSize</i>	Size of uncompressedData.

Returns

Compressed buffer.

Exceptions

<i>Error::StrategyError</i>	Error in compression unit.
-----------------------------	----------------------------

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

E.21.4.4 `virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (const Memory::uint8Array & uncompressedData) const throw (Error::StrategyError) [pure virtual]`

Compress a buffer.

Parameters

<i>uncompressed-Data</i>	Uncompressed data buffer to compress.
--------------------------	---------------------------------------

Returns

Compressed buffer.

Exceptions

Error::StrategyError	Error in decompression unit.
--------------------------------------	------------------------------

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

E.21.4.5 `virtual void BiometricEvaluation::IO::Compressor::compress (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize, const string & outputFile) const throw (Error::ObjectExists, Error::StrategyError) [pure virtual]`

Compress a buffer.

Parameters

<i>uncompressed-Data</i>	Uncompressed data buffer to compress.
<i>uncompressed-DataSize</i>	Size of uncompressedData.
<i>outputFile</i>	Location to save compressed file.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

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

E.21.4.6 `virtual void BiometricEvaluation::IO::Compressor::compress (const Memory::uint8Array & uncompressedData, const string & outputFile) const throw (Error::ObjectExists, Error::StrategyError) [pure virtual]`

Compress a buffer.

Parameters

<i>uncompressed-Data</i>	Uncompressed data buffer to compress.
<i>outputFile</i>	Location to save compressed file.

Exceptions

<i>Error::ObjectExists</i>	Output file already exists.
<i>Error::StrategyError</i>	Error in decompression unit.

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

E.21.4.7 `virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (const string & inputFile) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]`

Compress a file.

Parameters

<i>inputFile</i>	Path to file to compress.
------------------	---------------------------

Returns

Compressed buffer.

Exceptions

<i>Error::ObjectDoesNotExist</i>	Input file does not exist.
<i>Error::StrategyError</i>	Error in decompression unit.

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

E.21.4.8 `virtual void BiometricEvaluation::IO::Compressor::compress (const string & inputFile, const string & outputFile) const throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError) [pure virtual]`

Compress a file.

Parameters

<i>inputFile</i>	Path to file to compress.
<i>outputFile</i>	Path to location where compressed version will be saved.

Exceptions

<i>Error::ObjectDoesNotExist</i>	Input file does not exist.
<i>Error::ObjectExists</i>	Output file already exists.
<i>Error::StrategyError</i>	Error in decompression unit.

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

E.21.4.9 `virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (const uint8_t *const compressedData, uint64_t compressedDataSize) const throw (Error::StrategyError) [pure virtual]`

Decompress a compressed buffer.

Parameters

<i>compressed-Data</i>	Compressed data buffer to decompress.
<i>compressed-DataSize</i>	Size of compressedData.

Returns

Decompressed buffer.

Exceptions

<i>Error::StrategyError</i>	Error in compression unit.
---	----------------------------

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

E.21.4.10 `virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (const Memory::uint8Array & compressedData) const throw (Error::StrategyError) [pure virtual]`

Decompress a compressed buffer.

Parameters

<i>compressed-Data</i>	Compressed data buffer to decompress.
------------------------	---------------------------------------

Returns

Decompressed buffer.

Exceptions

<i>Error::StrategyError</i>	Error in decompression unit.
---	------------------------------

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

E.21.4.11 `virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (const string & inputFile) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]`

Decompress a compressed buffer into a file.

Parameters

<i>inputFile</i>	Location to save compressed file.
------------------	-----------------------------------

Returns

Decompressed buffer.

Exceptions

<i>Error::StrategyError</i>	Error in decompression unit.
<i>Error::ObjectDoesNotExist</i>	Output file already exists.

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

E.21.4.12 `virtual void BiometricEvaluation::IO::Compressor::decompress (const Memory::uint8Array & compressedData, const string & outputFile) const throw (Error::ObjectExists, Error::StrategyError) [pure virtual]`

Decompress a file.

Parameters

<i>compressed-Data</i>	Compressed data buffer to decompress.
<i>outputFile</i>	Path to location where decompressed version will be saved.

Exceptions

<i>Error::ObjectExists</i>	Output file already exists.
<i>Error::StrategyError</i>	Error in compression unit.

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

E.21.4.13 `virtual void BiometricEvaluation::IO::Compressor::decompress (const uint8_t *const compressedData, const uint64_t compressedDataSize, const string & outputFile) const throw (Error::ObjectExists, Error::StrategyError) [pure virtual]`

Decompress a file.

Parameters

<i>compressed-Data</i>	Compressed data buffer to decompress.
<i>compressed-DataSize</i>	Size of compressedData.
<i>outputFile</i>	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

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

E.21.4.14 `virtual void BiometricEvaluation::IO::Compressor::decompress (const string & inputFile, const string & outputFile) const throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError) [pure virtual]`

Decompress a file.

Parameters

<i>inputFile</i>	Path to file to decompress.
<i>outputFile</i>	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectDoesNotExist	Input file does not exist.
Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

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

E.21.4.15 `void BiometricEvaluation::IO::Compressor::setOption (const string & optionName, const string & optionValue) throw (Error::StrategyError)`

Assign a compressor option.

Will overwrite existing values without warning.

Parameters

<i>optionName</i>	Name of the option to add.
<i>optionValue</i>	Value of the option.

Exceptions

<i>Error::StrategyError</i>	Error setting option.
---	---------------------------------------

E.21.4.16 void BiometricEvaluation::IO::Compressor::setOption (const string & *optionName*, int64_t *optionValue*) throw (Error::StrategyError)

Assign a compressor option.

Will overwrite existing values without warning.

Parameters

<i>optionName</i>	Name of the option to add.
<i>optionValue</i>	Value of the option.

Exceptions

<i>Error::StrategyError</i>	Error setting option.
---	---------------------------------------

E.21.4.17 string BiometricEvaluation::IO::Compressor::getOption (const string & *optionName*) const throw (Error::ObjectDoesNotExist)

Obtain a compressor option as an integer.

Parameters

<i>optionName</i>	Name of the option to obtain.
-------------------	-------------------------------

Returns

Value of compressor option.

E.21.4.18 int64_t BiometricEvaluation::IO::Compressor::getOptionAsInteger (const string & *optionName*) const throw (Error::ObjectDoesNotExist)

Obtain a compressor option as an integer.

Parameters

<i>optionName</i>	Name of the option to obtain.
-------------------	-------------------------------

Returns

Value of compressor option.

Exceptions

<i>Error::ObjectDoesNotExist</i>	The option was never set.
--	---------------------------

E.21.4.19 void BiometricEvaluation::IO::Compressor::removeOption (const string & *optionName*) throw (Error::ObjectDoesNotExist)

Remove a compressor option.

Parameters

<i>optionName</i>	Name of the option to remove.
-------------------	-------------------------------

E.21.4.20 static tr1::shared_ptr<Compressor> BiometricEvaluation::IO::Compressor::createCompressor (Compressor::Kind *compressorKind* = Compressor::GZIP) throw (Error::ObjectDoesNotExist) [static]

[Compressor](#) factory.

Parameters

<i>compressorKind</i>	A known kind of compressor.
-----------------------	-----------------------------

Returns

A new compressor with default options.

Exceptions

<i>Error::ObjectDoesNotExist</i>	Invalid compressor type.
--	--------------------------

E.21.5 Member Data Documentation

E.21.5.1 const string BiometricEvaluation::IO::Compressor::GZIPTYPE [static]

String representations of the compressors

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

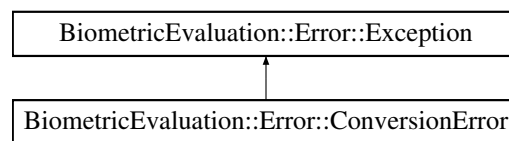
- be_io_compressor.h

E.22 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.22.1 Detailed Description

[Error](#) when converting one object into another, a property value from string to int, for example.

E.22.2 Constructor & Destructor Documentation

E.22.2.1 BiometricEvaluation::Error::ConversionError::ConversionError ()

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

Returns

The [ConversionError](#) object.

E.22.2.2 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.23 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.23.1 Detailed Description

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

E.23.2 Constructor & Destructor Documentation

E.23.2.1 [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	x	X-coordinate
in	y	Y-coordinate
in	xDistance	X-coordinate distance from origin
in	yDistance	Y-coordinate distance from origin

E.23.3 Member Data Documentation

E.23.3.1 [uint32_t BiometricEvaluation::Image::Coordinate::x](#)

X-coordinate

E.23.3.2 [uint32_t BiometricEvaluation::Image::Coordinate::y](#)

Y-coordinate

E.23.3.3 float BiometricEvaluation::Image::Coordinate::xDistance

X-coordinate distance from origin

E.23.3.4 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.24 BiometricEvaluation::Feature::CorePoint Struct Reference

Representation of the core.

```
#include <be_feature_minutiae.h>
```

Public Member Functions

- [CorePoint](#) ([Image::Coordinate](#) coordinate, bool has_angle=false, int angle=0)
Create a [CorePoint](#) struct.

Public Attributes

- [Image::Coordinate](#) **coordinate**
- bool **has_angle**
- int **angle**

E.24.1 Detailed Description

Representation of the core.

A core has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

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

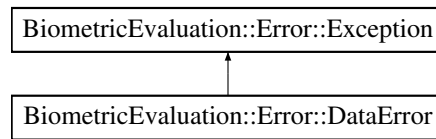
- `be_feature_minutiae.h`

E.25 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.25.1 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.25.2 Constructor & Destructor Documentation

E.25.2.1 [BiometricEvaluation::Error::DataError::DataError](#) ()

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

Returns

The [DataError](#) object.

E.25.2.2 [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:

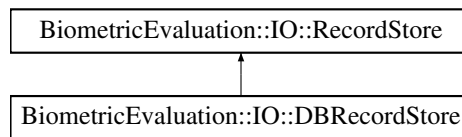
- `be_error_exception.h`

E.26 [BiometricEvaluation::IO::DBRecordStore](#) Class Reference

A class that implements [IO::RecordStore](#) using a Berkeley DB database as the underlying record storage system.

```
#include <be_io_dbrecstore.h>
```


Inheritance diagram for BiometricEvaluation::IO::DBRecordStore:



Public Member Functions

- [DBRecordStore](#) (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- [DBRecordStore](#) (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 [sync](#) () const throw (Error::StrategyError)
- 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)
- void [replace](#) (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t [length](#) (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void [flush](#) (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t [sequence](#) (string &key, void *const data=NULL, int cursor=[BE_RECSTORE_SEQ_NEXT](#)) throw (Error::ObjectDoesNotExist, Error::StrategyError)
Sequence through a [RecordStore](#), returning the key/data pairs.
- void [setCursorAtKey](#) (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void [changeName](#) (const string &name) throw (Error::ObjectExists, Error::StrategyError)

E.26.1 Detailed Description

A class that implements [IO::RecordStore](#) using a Berkeley DB database as the underlying record storage system.

E.26.2 Constructor & Destructor Documentation

E.26.2.1 BiometricEvaluation::IO::DBRecordStore::DBRecordStore (const string & name, const string & description, const string & parentDir) throw (Error::ObjectExists, Error::StrategyError)

Create a new [DBRecordStore](#), 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

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

E.26.2.2 `BiometricEvaluation::IO::DBRecordStore::DBRecordStore (const string & name, const string & parentDir, uint8_t mode = IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)`

Open an existing [DBRecordStore](#).

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

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

E.26.3 Member Function Documentation

E.26.3.1 `uint64_t BiometricEvaluation::IO::DBRecordStore::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

<i>Error::StrategyError</i>	An error occurred when using the underlying storage system.
---	---

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.26.3.2 `void BiometricEvaluation::IO::DBRecordStore::sync () const throw (Error::StrategyError) [virtual]`

Synchronize the entire record store to persistent storage.

Exceptions

<i>Error::StrategyError</i>	An error occurred when using the underlying storage system.
---	---

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.26.3.3 void BiometricEvaluation::IO::DBRecordStore::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

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.26.3.4 void BiometricEvaluation::IO::DBRecordStore::remove (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Remove a record from the store.

Parameters

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

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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.26.3.5 uint64_t BiometricEvaluation::IO::DBRecordStore::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	<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

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.26.3.6 `void BiometricEvaluation::IO::DBRecordStore::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	<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.26.3.7 `uint64_t BiometricEvaluation::IO::DBRecordStore::length (const string & key) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.26.3.8 `void BiometricEvaluation::IO::DBRecordStore::flush (const string & key) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.26.3.9 `uint64_t BiometricEvaluation::IO::DBRecordStore::sequence (string & key, void *const data = NULL, int cursor = BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Sequence through a [RecordStore](#), returning the key/data pairs.

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

Parameters

out	<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

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.26.3.10 void BiometricEvaluation::IO::DBRecordStore::setCursorAtKey (string & key) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

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

Parameters

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

Exceptions

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.26.3.11 void BiometricEvaluation::IO::DBRecordStore::changeName (const string & name) throw (Error::ObjectExists, Error::StrategyError) [virtual]

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

Parameters

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

Exceptions

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

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

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

- be_io_dbrecstore.h

E.27 BiometricEvaluation::Feature::DeltaPoint Struct Reference

Representation of the delta.

```
#include <be_feature_minutiae.h>
```

Public Member Functions

- [DeltaPoint](#) ([Image::Coordinate](#) coordinate, bool has_angle=false, int angle1=0, int angle2=0, int angle3=0)

Create a *DeltaPoint* struct.

Public Attributes

- *Image::Coordinate* **coordinate**
- bool **has_angle**
- int **angle1**
- int **angle2**
- int **angle3**

E.27.1 Detailed Description

Representation of the delta.

A delta has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

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

- *be_feature_minutiae.h*

E.28 BiometricEvaluation::View::AN2KView::DeviceMonitoringMode Class Reference

The level of human monitoring for the image capture device.

```
#include <be_view_an2kview.h>
```

Public Types

- enum *Kind* { *Controlled*, *Assisted*, *Observed*, *Unattended*, *Unknown*, *NA* }

E.28.1 Detailed Description

The level of human monitoring for the image capture device.

E.28.2 Member Enumeration Documentation

E.28.2.1 enum BiometricEvaluation::View::AN2KView::DeviceMonitoringMode::Kind

Enumerator:

Controlled Operator physically controls the subject to acquire biometric sample.

Assisted Person available to provide assistance to the subject submitting the biometric.

Observed Person present to observe the operation of the device but provides no assistance.

Unattended No one present to observe or provide assistance.

Unknown No information is known.

NA Optional field -- not specified

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

- `be_view_an2kview.h`

E.29 BiometricEvaluation::DataInterchange::AN2KRecord::DomainName Struct Reference

Representation of a domain name for the user-defined Type-2 logical record implementation.

```
#include <be_data_interchange_an2k.h>
```

Public Member Functions

- [DomainName](#) (string `identifier`="", string `version`="")
Create a [DomainName](#) struct.

Public Attributes

- string `identifier`
- string `version`

E.29.1 Detailed Description

Representation of a domain name for the user-defined Type-2 logical record implementation.

E.29.2 Constructor & Destructor Documentation

E.29.2.1 `BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::DomainName (string identifier = "", string version = " ") [inline]`

Create a [DomainName](#) struct.

Parameters

<i>identifier</i>	Unique identifier for agency, entity, or implementation.
<i>version</i>	Optional unique version number of the implementation of the identifier.

E.29.3 Member Data Documentation

E.29.3.1 `string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::identifier`

Unique identifier for agency, entity, or implementation.

E.29.3.2 string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::version

Optional version of the implementation

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

- `be_data_interchange_an2k.h`

E.30 BiometricEvaluation::Feature::AN2K7Minutiae::EncodingMethod Class - Reference

Methods for encoding minutiae data in an AN2K record.

```
#include <be_feature_an2k7minutiae.h>
```

Public Types

- enum **Kind** { **Automatic** = 0, **AutomaticUnedited**, **AutomaticEdited**, **Manual** }

E.30.1 Detailed Description

Methods for encoding minutiae data in an AN2K record.

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

- `be_feature_an2k7minutiae.h`

E.31 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry Struct Reference

Public Member Functions

- [Entry](#) (bool [standard](#), std::string [code](#))

Public Attributes

- bool [standard](#)
- std::string [code](#)

E.31.1 Constructor & Destructor Documentation

E.31.1.1 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::Entry (bool *standard*, std::string *code*)

Create an [Entry](#) struct.

Parameters

<i>standard</i>	Whether or not code is a standard AN2K pattern classification code.
<i>code</i>	AN2K or user-defined pattern classification code.

E.31.2 Member Data Documentation**E.31.2.1 `bool BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::standard`**

Whether code is a standard AN2K pattern classification code.

E.31.2.2 `std::string BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::code`

AN2K or user-defined pattern classification code.

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

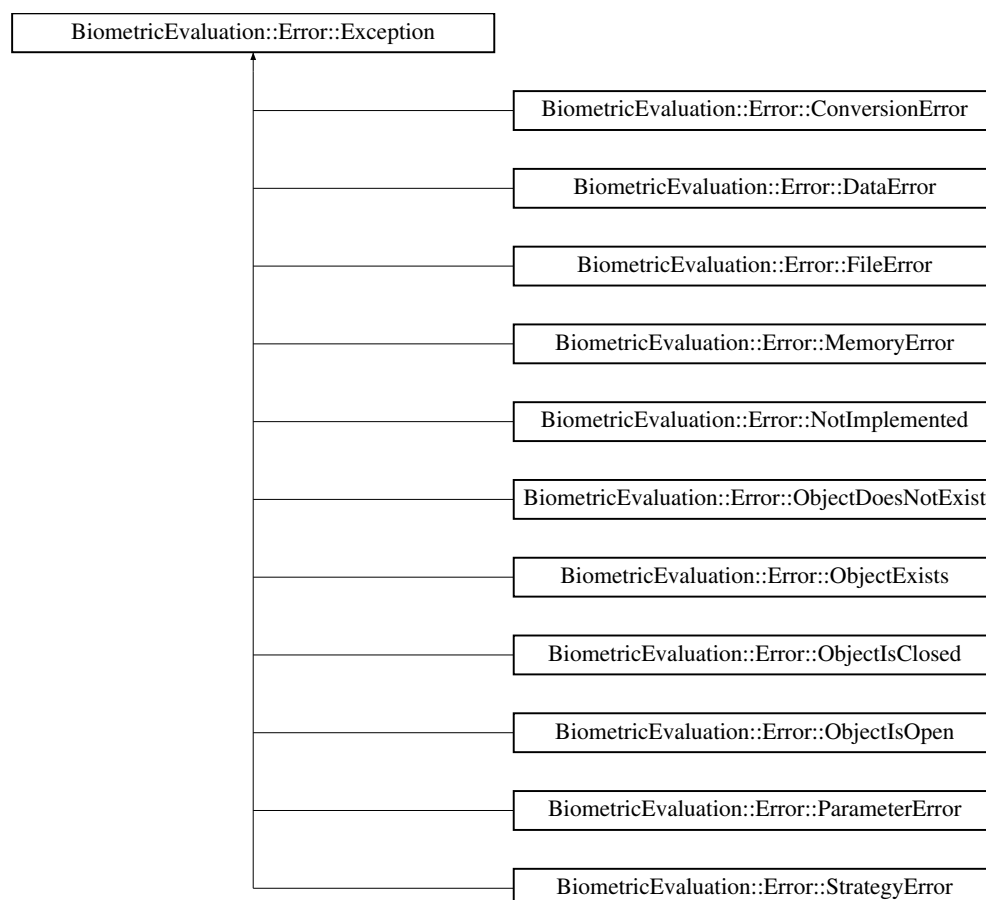
- `be_feature_an2k7minutiae.h`

E.32 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.32.1 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.32.2 Constructor & Destructor Documentation

E.32.2.1 BiometricEvaluation::Error::Exception::Exception ()

Construct an [Exception](#) object without an information string.

Returns

The [Exception](#) object.

E.32.2.2 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.32.3 Member Function Documentation**E.32.3.1 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:

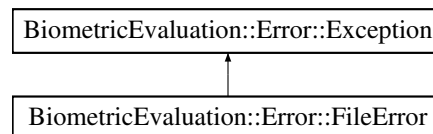
- `be_error_exception.h`

E.33 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.33.1 Detailed Description

File error when opening, reading, writing, etc.

E.33.2 Constructor & Destructor Documentation

E.33.2.1 BiometricEvaluation::Error::FileError::FileError ()

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

Returns

The [FileError](#) object.

E.33.2.2 BiometricEvaluation::Error::FileError::FileError (string *info*)

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

Returns

The [FileError](#) object.

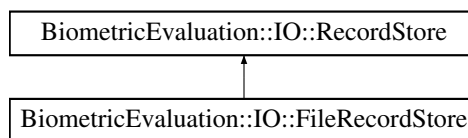
The documentation for this class was generated from the following file:

- `be_error_exception.h`

E.34 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)
Sequence through a [RecordStore](#), returning the key/data pairs.
- void [setCursorAtKey](#) (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void [changeName](#) (const string &name) throw (Error::ObjectExists, Error::StrategyError)

Protected Member Functions

- string [canonicalName](#) (const string &name) const

E.34.1 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.34.2 Constructor & Destructor Documentation

E.34.2.1 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

Error::ObjectExists	The store already exists.
Error::StrategyError	An error occurred when accessing the underlying file system.

E.34.2.2 `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

Error::ObjectDoesNotExist	The store does not exist.
Error::StrategyError	An error occurred when accessing the underlying file system.

E.34.3 Member Function Documentation

E.34.3.1 `uint64_t BiometricEvaluation::IO::FileRecordStore::getSpaceUsed () const throw (Error::StrategyError) [virtual]`

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the [RecordStore](#).

Exceptions

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

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.34.3.2 `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

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.34.3.3 `void BiometricEvaluation::IO::FileRecordStore::remove (const string & key) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Remove a record from the store.

Parameters

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

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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.34.3.4 `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	<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

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.34.3.5 `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	<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.34.3.6 `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	<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.34.3.7 `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

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.34.3.8 `uint64_t BiometricEvaluation::IO::FileRecordStore::sequence (string & key, void *const data = NULL, int cursor = BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Sequence through a [RecordStore](#), returning the key/data pairs.

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

Parameters

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

Returns

The length of the record currently in sequence.

Exceptions

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.34.3.9 `void BiometricEvaluation::IO::FileRecordStore::setCursorAtKey (string & key) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

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

Parameters

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

Exceptions

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.34.3.10 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 RecordStore .
----	-------------	--

Exceptions

<i>Error::StrategyError</i>	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.35 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.35.1 Detailed Description

Joint and tip codes.

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

- be_finger.h

E.36 BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem Struct Reference

Representation of information about a fingerprint reader system.

```
#include <be_feature_an2k7minutiae.h>
```

Public Attributes

- string [name](#)
- EncodingMethod::Kind [method](#)
- string [equipment](#)

E.36.1 Detailed Description

Representation of information about a fingerprint reader system.

E.36.2 Member Data Documentation

E.36.2.1 string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::name

Name for system that encoded minutiae

E.36.2.2 EncodingMethod::Kind BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::method

Method used to encoded minutiae

E.36.2.3 string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::equipment

Optional ID for equipment used in system

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

- [be_feature_an2k7minutiae.h](#)

E.37 BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition Struct Reference

Locations of an individual finger segment in a slap.

```
#include <be_finger_an2kview_capture.h>
```

Public Member Functions

- [FingerSegmentPosition](#) (const Finger::Position::Kind [fingerPosition](#), const Image::CoordinateSet [coordinates](#))

Create an [FingerSegmentPosition](#) struct.

Public Attributes

- Finger::Position::Kind [fingerPosition](#)
- Image::CoordinateSet [coordinates](#)

E.37.1 Detailed Description

Locations of an individual finger segment in a slap.

E.37.2 Constructor & Destructor Documentation

- E.37.2.1** [BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::FingerSegmentPosition](#) (const Finger::Position::Kind [fingerPosition](#), const Image::CoordinateSet [coordinates](#))

Create an [FingerSegmentPosition](#) struct.

Parameters

<i>fingerPosition</i>	Finger depicted in this segment.
<i>coordinates</i>	Collection of coordinates that compose the segment bonding polygon.

E.37.3 Member Data Documentation

- E.37.3.1** [Finger::Position::Kind](#) [BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::fingerPosition](#)

[Finger](#) depicted in this segment

- E.37.3.2** [Image::CoordinateSet](#) [BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::coordinates](#)

Points composing the segmented polygon

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

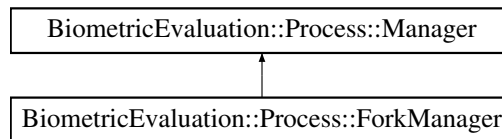
- `be_finger_an2kview_capture.h`

E.38 BiometricEvaluation::Process::ForkManager Class Reference

[Manager](#) implementation that starts Workers by calling `fork(2)`.

```
#include <be_process_forkmanager.h>
```

Inheritance diagram for BiometricEvaluation::Process::ForkManager:



Public Member Functions

- [ForkManager](#) ()
- uint32_t [getNumCompletedWorkers](#) () const throw (Error::StrategyError)
Obtain the number of Workers that have exited.
- uint32_t [getNumActiveWorkers](#) () const throw (Error::StrategyError)
Obtain the number of Workers that are still running.
- uint32_t [getTotalWorkers](#) () const
Obtain the number of Workers this class is handling.
- tr1::shared_ptr< [WorkerController](#) > [addWorker](#) (tr1::shared_ptr< [Worker](#) > worker)
Adds a [Worker](#) to be managed by this [Manager](#).
- void [startWorkers](#) (bool wait=true) throw (Error::ObjectExists, Error::StrategyError)
Begin Worker's work.
- void [startWorker](#) (tr1::shared_ptr< [WorkerController](#) > worker, bool wait=true) throw (Error::ObjectExists, Error::StrategyError)
Start a worker.
- int32_t [stopWorker](#) (tr1::shared_ptr< [WorkerController](#) > workerController) throw (Error::ObjectDoesNotExist, Error::StrategyError)
Ask [Worker](#) to exit.
- void [reset](#) () throw (Error::ObjectExists)
Reuse all Workers.
- [~ForkManager](#) ()
[ForkManager](#) destructor.

Protected Attributes

- vector< tr1::shared_ptr < [ForkWorkerController](#) > > [_workers](#)

E.38.1 Detailed Description

[Manager](#) implementation that starts Workers by calling fork(2).

E.38.2 Constructor & Destructor Documentation

E.38.2.1 BiometricEvaluation::Process::ForkManager::ForkManager ()

[ForkManager](#) constructor.

E.38.3 Member Function Documentation

E.38.3.1 `uint32_t BiometricEvaluation::Process::ForkManager::getNumCompletedWorkers () const throw (Error::StrategyError) [virtual]`

Obtain the number of Workers that have exited.

Returns

The number of Workers that have exited.

Exceptions

<i>Error::StrategyError</i>	No Workers have started working yet.
---	--------------------------------------

Implements [BiometricEvaluation::Process::Manager](#).

E.38.3.2 `uint32_t BiometricEvaluation::Process::ForkManager::getNumActiveWorkers () const throw (Error::StrategyError) [virtual]`

Obtain the number of Workers that are still running.

Returns

The number of Workers that are still running.

Exceptions

<i>Error::StrategyError</i>	No Workers have started Working yet.
---	--------------------------------------

Implements [BiometricEvaluation::Process::Manager](#).

E.38.3.3 `uint32_t BiometricEvaluation::Process::ForkManager::getTotalWorkers () const [virtual]`

Obtain the number of Workers this class is handling.

Returns

Number of Workers.

Implements [BiometricEvaluation::Process::Manager](#).

E.38.3.4 `tr1::shared_ptr<WorkerController> BiometricEvaluation::Process::ForkManager::addWorker (tr1::shared_ptr< Worker > worker) [virtual]`

Adds a [Worker](#) to be managed by this [Manager](#).

Parameters

<i>worker</i>	A Worker instance to run.
---------------	---

Returns

shared_ptr to worker.

Implements [BiometricEvaluation::Process::Manager](#).

E.38.3.5 void [BiometricEvaluation::Process::ForkManager::startWorkers](#) (bool *wait* = true) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Begin Worker's work.

Parameters

<i>in</i>	<i>wait</i>	Whether or not to wait for all Workers to return before returning.
-----------	-------------	--

Exceptions

Error::ObjectExists	One or more of the Workers is already working.
Error::StrategyError	Problem forking.

Implements [BiometricEvaluation::Process::Manager](#).

E.38.3.6 void [BiometricEvaluation::Process::ForkManager::startWorker](#) (tr1::shared_ptr< WorkerController > *worker*, bool *wait* = true) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Start a worker.

Parameters

<i>worker</i>	Pointer to a WorkerController that is being managed by this Manager instance.
<i>wait</i>	Whether or not to wait for this Worker to exit before returning control to the caller.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

Implements [BiometricEvaluation::Process::Manager](#).

E.38.3.7 int32_t [BiometricEvaluation::Process::ForkManager::stopWorker](#) (tr1::shared_ptr< WorkerController > *workerController*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Ask [Worker](#) to exit.

Sends SIGUSR1 to the [Worker](#), which [ForkManager](#) will handle automatically.

Parameters

<i>worker-Controller</i>	Pointer to the ForkWorkerController that should be stopped.
--------------------------	---

Returns

Exit status of worker.

Exceptions

<i>Error::ObjectDoesNotExist</i>	worker is not working.
<i>Error::StrategyError</i>	Problem sending the signal.

Implements [BiometricEvaluation::Process::Manager](#).

E.38.3.8 `void BiometricEvaluation::Process::ForkManager::reset () throw (Error::ObjectExists)`
[virtual]

Reuse all Workers.

Exceptions

<i>Error::ObjectExists</i>	At least one Worker is still working.
--	---

Implements [BiometricEvaluation::Process::Manager](#).

E.38.4 Member Data Documentation

E.38.4.1 `vector<tr1::shared_ptr<ForkWorkerController> > BiometricEvaluation::Process::ForkManager::_workers` [protected]

Workers that have been added

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

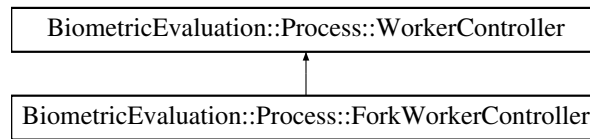
- `be_process_forkmanager.h`

E.39 BiometricEvaluation::Process::ForkWorkerController Class Reference

Wrapper of a [Worker](#) returned from a [Process::ForkManager](#).

```
#include <be_process_forkmanager.h>
```

Inheritance diagram for BiometricEvaluation::Process::ForkWorkerController:



Public Member Functions

- `bool isWorking () const`
Obtain whether or not [Worker](#) is working.
- `void reset () throw (Error::ObjectExists)`
Reuse the [Worker](#).
- `pid_t getPID () const`
Obtain the PID of this process this instance represents.
- `~ForkWorkerController ()`
[ForkWorkerController](#) destructor.

Static Public Member Functions

- `static void _stop (int signal)`
Tell [_this](#) to stop.

Friends

- `void ForkManager::startWorkers (bool wait) throw (Error::ObjectExists, Error::StrategyError)`
Begin [Worker](#)'s work.
- `void ForkManager::startWorker (tr1::shared_ptr< WorkerController > worker, bool wait) throw (Error::ObjectExists, Error::StrategyError)`
Restart a completed [Worker](#).
- `int32_t ForkManager::stopWorker (tr1::shared_ptr< WorkerController > workerController) throw (-Error::ObjectDoesNotExist, Error::StrategyError)`
Ask [Worker](#) to exit.
- `tr1::shared_ptr< WorkerController > ForkManager::addWorker (tr1::shared_ptr< Worker > worker)`
Adds a [Worker](#) to be managed by this [Manager](#).

E.39.1 Detailed Description

Wrapper of a [Worker](#) returned from a [Process::ForkManager](#).

E.39.2 Member Function Documentation

E.39.2.1 `bool BiometricEvaluation::Process::ForkWorkerController::isWorking () const` [virtual]

Obtain whether or not [Worker](#) is working.

Returns

Whether or not the [Worker](#) is working.

Implements [BiometricEvaluation::Process::WorkerController](#).

E.39.2.2 `void BiometricEvaluation::Process::ForkWorkerController::reset () throw (Error::ObjectExists)`
[virtual]

Reuse the [Worker](#).

Exceptions

Error::ObjectExists	The previously started Worker is still running.
-------------------------------------	---

Reimplemented from [BiometricEvaluation::Process::WorkerController](#).

E.39.2.3 `pid_t BiometricEvaluation::Process::ForkWorkerController::getPID () const`

Obtain the PID of this process this instance represents.

Returns

pid of the process this instance represents.

Note

Call `isRunning()` before doing anything with the PID returned from this function.

E.39.2.4 `static void BiometricEvaluation::Process::ForkWorkerController::_stop (int signal)` [static]

Tell `_this` to stop.

Called by the child process instance when SIGUSR1 is received.

Parameters

<i>signal</i>	The signal caught that prompted this function to be called (SIGUSR1).
---------------	---

E.39.3 Friends And Related Function Documentation

E.39.3.1 `void ForkManager::startWorkers (bool wait) throw (Error::ObjectExists, Error::StrategyError)`
[friend]

Begin Worker's work.

Parameters

<i>in</i>	<i>wait</i>	Whether or not to wait for all Workers to return before returning.
-----------	-------------	--

Exceptions

<i>Error::ObjectExists</i>	One or more of the Workers is already working.
<i>Error::StrategyError</i>	Problem forking.

E.39.3.2 `void ForkManager::startWorker (tr1::shared_ptr< WorkerController > worker, bool wait) throw (Error::ObjectExists, Error::StrategyError) [friend]`

Restart a completed [Worker](#).

Parameters

<i>worker</i>	Pointer to a WorkerController that is being managed by this Manager instance.
<i>wait</i>	Whether or not to wait for this Worker to exit before returning control to the caller.

Exceptions

<i>Error::ObjectExists</i>	worker is already working.
<i>Error::StrategyError</i>	worker is not managed by this Manager instance.

E.39.3.3 `int32_t ForkManager::stopWorker (tr1::shared_ptr< WorkerController > workerController) throw (Error::ObjectDoesNotExist, Error::StrategyError) [friend]`

Ask [Worker](#) to exit.

Sends SIGUSR1 to the [Worker](#), which [ForkManager](#) will handle automatically.

Parameters

<i>worker-Controller</i>	Pointer to the ForkWorkerController that should be stopped.
--------------------------	---

Returns

Exit status of worker.

Exceptions

<i>Error::ObjectDoesNotExist</i>	worker is not working.
<i>Error::StrategyError</i>	Problem sending the signal.

E.39.3.4 `tr1::shared_ptr<WorkerController> ForkManager::addWorker (tr1::shared_ptr< Worker > worker) [friend]`

Adds a [Worker](#) to be managed by this [Manager](#).

Parameters

<i>worker</i>	A Worker instance to run.
---------------	---

Returns

`shared_ptr` to `worker`.

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

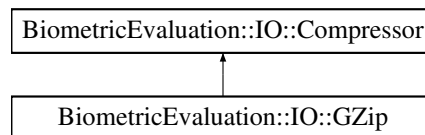
- `be_process_forkmanager.h`

E.40 BiometricEvaluation::IO::GZip Class Reference

[Compressor](#) for gzip compression from zlib.

```
#include <be_io_gzip.h>
```

Inheritance diagram for `BiometricEvaluation::IO::GZip`:



Public Member Functions

- [Memory::uint8Array compress](#) (const `uint8_t` *const uncompressedData, `uint64_t` uncompressedDataSize) const throw (Error::StrategyError)
Compress a buffer.
- [Memory::uint8Array compress](#) (const [Memory::uint8Array](#) &uncompressedData) const throw (Error::StrategyError)
Compress a buffer.
- void [compress](#) (const `uint8_t` *const uncompressedData, `uint64_t` uncompressedDataSize, const string &outputFile) const throw (Error::ObjectExists, Error::StrategyError)
Compress a buffer.
- void [compress](#) (const [Memory::uint8Array](#) &uncompressedData, const string &outputFile) const throw (Error::ObjectExists, Error::StrategyError)
Compress a buffer.
- [Memory::uint8Array compress](#) (const string &inputFile) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
Compress a file.

- void [compress](#) (const string &inputFile, const string &outputFile) const throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError)
Compress a file.
- [Memory::uint8Array decompress](#) (const uint8_t *const compressedData, uint64_t compressedDataSize) const throw (Error::StrategyError)
Decompress a compressed buffer.
- [Memory::uint8Array decompress](#) (const [Memory::uint8Array](#) &compressedData) const throw (Error::StrategyError)
Decompress a compressed buffer.
- [Memory::uint8Array decompress](#) (const string &input) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
Decompress a compressed buffer into a file.
- void [decompress](#) (const string &inputFile, const string &outputFile) const throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError)
Decompress a file.
- void [decompress](#) (const uint8_t *const compressedData, const uint64_t compressedDataSize, const string &outputFile) const throw (Error::ObjectExists, Error::StrategyError)
Decompress a file.
- void [decompress](#) (const [Memory::uint8Array](#) &compressedData, const string &outputFile) const throw (Error::ObjectExists, Error::StrategyError)
Decompress a file.

Static Public Attributes

- static const string [COMPRESSION_LEVEL](#)
- static const string [COMPRESSION_STRATEGY](#)
- static const string [COMPRESSION_METHOD](#)
- static const string [INPUT_DATA_TYPE](#)
- static const string [WINDOW_BITS](#)
- static const string [MEMORY_LEVEL](#)
- static const string [CHUNK_SIZE](#)

E.40.1 Detailed Description

[Compressor](#) for gzip compression from zlib.

E.40.2 Member Function Documentation

- E.40.2.1** [Memory::uint8Array BiometricEvaluation::IO::GZip::compress](#) (const uint8_t *const *uncompressedData*, uint64_t *uncompressedDataSize*) const throw (Error::StrategyError)
[virtual]

Compress a buffer.

Parameters

<i>uncompressed-Data</i>	Uncompressed data buffer to compress.
<i>uncompressed-DataSize</i>	Size of uncompressedData.

Returns

Compressed buffer.

Exceptions

<i>Error::StrategyError</i>	Error in compression unit.
---	--

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.2 `Memory::uint8Array BiometricEvaluation::IO::GZip::compress (const Memory::uint8Array & uncompressedData) const throw (Error::StrategyError) [virtual]`

Compress a buffer.

Parameters

<i>uncompressed-Data</i>	Uncompressed data buffer to compress.
--------------------------	---------------------------------------

Returns

Compressed buffer.

Exceptions

<i>Error::StrategyError</i>	Error in decompression unit.
---	--

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.3 `void BiometricEvaluation::IO::GZip::compress (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize, const string & outputFile) const throw (Error::ObjectExists, Error::StrategyError) [virtual]`

Compress a buffer.

Parameters

<i>uncompressed-Data</i>	Uncompressed data buffer to compress.
<i>uncompressed-DataSize</i>	Size of uncompressedData.
<i>outputFile</i>	Location to save compressed file.

Exceptions

<i>Error::ObjectExists</i>	Output file already exists.
<i>Error::StrategyError</i>	Error in compression unit.

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.4 `void BiometricEvaluation::IO::GZip::compress (const Memory::uint8Array & uncompressedData,
const string & outputFile) const throw (Error::ObjectExists, Error::StrategyError)
[virtual]`

Compress a buffer.

Parameters

<i>uncompressed-Data</i>	Uncompressed data buffer to compress.
<i>outputFile</i>	Location to save compressed file.

Exceptions

<i>Error::ObjectExists</i>	Output file already exists.
<i>Error::StrategyError</i>	Error in decompression unit.

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.5 `Memory::uint8Array BiometricEvaluation::IO::GZip::compress (const string & inputFile) const
throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Compress a file.

Parameters

<i>inputFile</i>	Path to file to compress.
------------------	---------------------------

Returns

Compressed buffer.

Exceptions

<i>Error::ObjectDoesNotExist</i>	Input file does not exist.
<i>Error::StrategyError</i>	Error in decompression unit.

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.6 `void BiometricEvaluation::IO::GZip::compress (const string & inputFile, const string & outputFile) const throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError)`
[virtual]

Compress a file.

Parameters

<i>inputFile</i>	Path to file to compress.
<i>outputFile</i>	Path to location where compressed version will be saved.

Exceptions

<i>Error::ObjectDoesNotExist</i>	Input file does not exist.
<i>Error::ObjectExists</i>	Output file already exists.
<i>Error::StrategyError</i>	Error in decompression unit.

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.7 `Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (const uint8_t *const compressedData, uint64_t compressedDataSize) const throw (Error::StrategyError)`
[virtual]

Decompress a compressed buffer.

Parameters

<i>compressed-Data</i>	Compressed data buffer to decompress.
<i>compressed-DataSize</i>	Size of compressedData.

Returns

Decompressed buffer.

Exceptions

<i>Error::StrategyError</i>	Error in compression unit.
---	----------------------------

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.8 `Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (const Memory::uint8Array & compressedData) const throw (Error::StrategyError)` [virtual]

Decompress a compressed buffer.

Parameters

<i>compressed-Data</i>	Compressed data buffer to decompress.
------------------------	---------------------------------------

Returns

Decompressed buffer.

Exceptions

<i>Error::StrategyError</i>	Error in decompression unit.
---	------------------------------

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.9 `Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (const string & inputFile) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Decompress a compressed buffer into a file.

Parameters

<i>inputFile</i>	Location to save compressed file.
------------------	-----------------------------------

Returns

Decompressed buffer.

Exceptions

<i>Error::StrategyError</i>	Error in decompression unit.
<i>Error::ObjectDoesNotExist</i>	Output file already exists.

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.10 `void BiometricEvaluation::IO::GZip::decompress (const string & inputFile, const string & outputFile) const throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError) [virtual]`

Decompress a file.

Parameters

<i>inputFile</i>	Path to file to decompress.
<i>outputFile</i>	Path to location where decompressed version will be saved.

Exceptions

<i>Error::ObjectDoesNotExist</i>	Input file does not exist.
<i>Error::ObjectExists</i>	Output file already exists.
<i>Error::StrategyError</i>	Error in compression unit.

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.11 void BiometricEvaluation::IO::GZip::decompress (const uint8_t *const *compressedData*, const uint64_t *compressedDataSize*, const string & *outputFile*) const throw (Error::ObjectExists, Error::StrategyError) [virtual]

Decompress a file.

Parameters

<i>compressed-Data</i>	Compressed data buffer to decompress.
<i>compressed-DataSize</i>	Size of compressedData.
<i>outputFile</i>	Path to location where decompressed version will be saved.

Exceptions

<i>Error::ObjectExists</i>	Output file already exists.
<i>Error::StrategyError</i>	Error in compression unit.

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.2.12 void BiometricEvaluation::IO::GZip::decompress (const Memory::uint8Array & *compressedData*, const string & *outputFile*) const throw (Error::ObjectExists, Error::StrategyError) [virtual]

Decompress a file.

Parameters

<i>compressed-Data</i>	Compressed data buffer to decompress.
<i>outputFile</i>	Path to location where decompressed version will be saved.

Exceptions

<i>Error::ObjectExists</i>	Output file already exists.
<i>Error::StrategyError</i>	Error in compression unit.

Implements [BiometricEvaluation::IO::Compressor](#).

E.40.3 Member Data Documentation

E.40.3.1 `const string BiometricEvaluation::IO::GZip::COMPRESSION_LEVEL` [static]

How thorough the compression should be

E.40.3.2 `const string BiometricEvaluation::IO::GZip::COMPRESSION_STRATEGY` [static]

Which underlying algorithm to use

E.40.3.3 `const string BiometricEvaluation::IO::GZip::COMPRESSION_METHOD` [static]

Which underlying method in the compressor

E.40.3.4 `const string BiometricEvaluation::IO::GZip::INPUT_DATA_TYPE` [static]

The type of data being compressed

E.40.3.5 `const string BiometricEvaluation::IO::GZip::WINDOW_BITS` [static]

Window size

E.40.3.6 `const string BiometricEvaluation::IO::GZip::MEMORY_LEVEL` [static]

How much memory for internal compression state

E.40.3.7 `const string BiometricEvaluation::IO::GZip::CHUNK_SIZE` [static]

How many bytes to work at a time

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

- `be_io_gzip.h`

E.41 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 tr1::shared_ptr< [Image](#) > [openImage](#) (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
Determine the image type of a buffer of image data and create an [Image](#) object.
- static tr1::shared_ptr< [Image](#) > [openImage](#) (const [Memory::uint8Array](#) &data) throw (Error::DataError, Error::StrategyError)
Determine the image type of a buffer of image data and create an [Image](#) object.
- static tr1::shared_ptr< [Image](#) > [openImage](#) (const string &path) throw (Error::DataError, Error::ObjectDoesNotExist, Error::StrategyError)
Determine the image type of an image file and create an [Image](#) object.
- static CompressionAlgorithm::Kind [getCompressionAlgorithm](#) (const uint8_t *data, const uint64_t size)
Determine the compression algorithm of a buffer of image data.
- static CompressionAlgorithm::Kind [getCompressionAlgorithm](#) (const [Memory::uint8Array](#) &data)
Determine the compression algorithm of a buffer of image data.
- static CompressionAlgorithm::Kind [getCompressionAlgorithm](#) (const string &path) throw (Error::ObjectDoesNotExist, Error::StrategyError)
Determine the compression algorithm of a file.

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.41.1 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.41.2 Constructor & Destructor Documentation

E.41.2.1 [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

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

Exceptions

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

E.41.2.2 `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 CompressionAlgorithm of data.

Exceptions

<code>Error::DataError</code>	Error manipulating data.
<code>Error::StrategyError</code>	Error while creating Image .

E.41.3 Member Function Documentation

E.41.3.1 `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.41.3.2 `Resolution BiometricEvaluation::Image::Image::getResolution () const`

Accessor for the resolution of the image.

Returns

[Resolution](#) struct

E.41.3.3 `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.41.3.4 `virtual Memory::AutoArray<uint8_t> BiometricEvaluation::Image::Image::getRawData () const throw (Error::DataError) [pure virtual]`

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

Returns

[Raw](#) image data.

Exceptions

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

Implemented in [BiometricEvaluation::Image::NetPBM](#), [BiometricEvaluation::Image::JPEG](#), [BiometricEvaluation::Image::JPEG2000](#), [BiometricEvaluation::Image::PNG](#), [BiometricEvaluation::Image::JPEGL](#), [BiometricEvaluation::Image::Raw](#), and [BiometricEvaluation::Image::WSQ](#).

E.41.3.5 `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

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

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. *depth* adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implemented in [BiometricEvaluation::Image::NetPBM](#), [BiometricEvaluation::Image::JPEG2000](#), [BiometricEvaluation::Image::PNG](#), [BiometricEvaluation::Image::JPEG](#), [BiometricEvaluation::Image::Raw](#), [BiometricEvaluation::Image::WSQ](#), and [BiometricEvaluation::Image::JPEGL](#).

E.41.3.6 `Size BiometricEvaluation::Image::Image::getDimensions () const`

Accessor for the dimensions of the image in pixels.

Returns

[Coordinate](#) object containing dimensions in pixels.

E.41.3.7 `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.41.3.8 `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.41.3.9 `static tr1::shared_ptr<Image> BiometricEvaluation::Image::Image::openImage (const uint8_t * data, const uint64_t size) throw (Error::DataError, Error::StrategyError) [static]`

Determine the image type of a buffer of image data and create an [Image](#) object.

Parameters

<i>in</i>	<i>data</i>	The image data.
<i>in</i>	<i>size</i>	The size of the image data, in bytes.

Returns

[Image](#) representation of the input data buffer.

Exceptions

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

E.41.3.10 `static tr1::shared_ptr<Image> BiometricEvaluation::Image::Image::openImage (const Memory::uint8Array & data) throw (Error::DataError, Error::StrategyError) [static]`

Determine the image type of a buffer of image data and create an [Image](#) object.

Parameters

<i>in</i>	<i>data</i>	The image data.
-----------	-------------	-----------------

Returns

[Image](#) representation of the input data buffer.

Exceptions

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

E.41.3.11 `static tr1::shared_ptr<Image> BiometricEvaluation::Image::Image::openImage (const string & path) throw (Error::DataError, Error::ObjectDoesNotExist, Error::StrategyError) [static]`

Determine the image type of an image file and create an [Image](#) object.

Parameters

<i>in</i>	<i>path</i>	Path to image data.
-----------	-------------	---------------------

Returns

[Image](#) representation of the input data buffer.

Exceptions

Error::DataError	Error manipulating data.
Error::ObjectDoesNotExist	No file at specified path.
Error::StrategyError	Error while creating Image .

E.41.3.12 `static CompressionAlgorithm::Kind BiometricEvaluation::Image::Image::getCompressionAlgorithm (const uint8_t* data, const uint64_t size) [static]`

Determine the compression algorithm of a buffer of image data.

Parameters

<i>in</i>	<i>data</i>	The image data.
<i>in</i>	<i>size</i>	The size of the image data, in bytes.

Returns

Compression algorithm used in the buffer.

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation Framework is found.

E.41.3.13 `static CompressionAlgorithm::Kind BiometricEvaluation::Image::Image::getCompressionAlgorithm (const Memory::uint8Array & data) [static]`

Determine the compression algorithm of a buffer of image data.

Parameters

in	<i>data</i>	The image data.
----	-------------	-----------------

Returns

Compression algorithm used in the buffer.

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation Framework is found.

E.41.3.14 `static CompressionAlgorithm::Kind BiometricEvaluation::Image::Image::getCompressionAlgorithm (const string & path) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]`

Determine the compression algorithm of a file.

Parameters

in	<i>path</i>	Path to file.
----	-------------	---------------

Returns

Compression algorithm used in the file.

Exceptions

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

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation Framework is found.

E.41.3.15 `void BiometricEvaluation::Image::Image::setResolution (const Resolution resolution)`
 [protected]

Mutator for the resolution of the image .

Parameters

<code>in</code>	<code><i>resolution</i></code>	Resolution struct.
-----------------	--------------------------------	------------------------------------

E.41.3.16 `void BiometricEvaluation::Image::Image::setDimensions (const Size dimensions)`
 [protected]

Mutator for the dimensions of the image in pixels.

Parameters

<code>in</code>	<code><i>dimensions</i></code>	Dimensions of image (pixel).
-----------------	--------------------------------	------------------------------

E.41.3.17 `void BiometricEvaluation::Image::Image::setDepth (const uint32_t depth)` [protected]

Mutator for the color depth of the image in bits.

Parameters

<code>in</code>	<code><i>depth</i></code>	The color depth of the image (bit).
-----------------	---------------------------	-------------------------------------

E.41.4 Member Data Documentation

E.41.4.1 `const uint32_t BiometricEvaluation::Image::Image::bitsPerComponent = 8` [static]

Number of bits per color component

E.41.4.2 `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.42 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.42.1 Detailed Description

[Finger](#) and palm impression types.

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

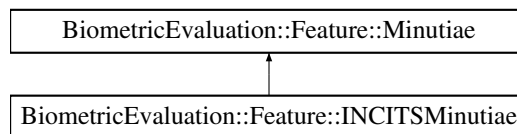
- `be_finger.h`

E.43 BiometricEvaluation::Feature::INCITSMinutiae Class Reference

A class to represent a set of minutiae in an ANSI/INCITS record.

```
#include <be_feature_incitsminutiae.h>
```

Inheritance diagram for BiometricEvaluation::Feature::INCITSMinutiae:



Public Member Functions

- MinutiaeFormat::Kind [getFormat](#) () const
Obtain the minutiae format kind.
- MinutiaPointSet [getMinutiaPoints](#) () const
Obtain the set of finger minutiae data points. The set may be empty.
- RidgeCountItemSet [getRidgeCountItems](#) () const
Obtain the set of ridge count data items. The set may be empty.
- CorePointSet [getCores](#) () const
Obtains the set of core positions. The set may be empty.
- DeltaPointSet [getDeltas](#) () const
Obtains the set of delta positions. The set may be empty.
- [INCITSMinutiae](#) (const MinutiaPointSet &mps, const RidgeCountItemSet &rcis, const CorePointSet &cps, const DeltaPointSet &dps)
Construct an INCITS [Minutiae](#) object from its components.

- [INCITSMinutiae](#) ()
Default constructor for an INCITS [Minutiae](#) object.
- void [setMinutiaPoints](#) (const MinutiaPointSet &mps)
Mutator for the minutiae point set.
- void [setRidgeCountItems](#) (const RidgeCountItemSet &rcis)
Mutator for the ridge count items.
- void [setCorePointSet](#) (const CorePointSet &cps)
Mutator for the set of core points.
- void [setDeltaPointSet](#) (const DeltaPointSet &dps)
Mutator for the set of delta points.

Static Public Attributes

- static const string **FMR_ANSI_SPEC_VERSION**
- static const string **FMR_ISO_SPEC_VERSION**
- static const string **FMR_ANSI07_SPEC_VERSION**
- static const uint8_t **FMR_SPEC_VERSION_LEN** = 4
- static const uint32_t **FED_HEADER_LENGTH** = 4
- static const uint32_t **FED_RCD_ITEM_LENGTH** = 3
- static const uint16_t **FMD_MINUTIA_TYPE_MASK** = 0xC000
- static const uint16_t **FMD_RESERVED_MASK** = 0xC000
- static const uint16_t **FMD_MINUTIA_TYPE_SHIFT** = 14
- static const uint16_t **FMD_RESERVED_SHIFT** = 14
- static const uint16_t **FMD_X_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_Y_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_MASK** = 0xC0
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_SHIFT** = 6
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_ANGLE_MASK** = 0x3F
- static const uint16_t **FMD_MIN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MAX_MINUTIA_QUALITY** = 100
- static const uint16_t **FMD_UNKNOWN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MIN_MINUTIA_ANGLE** = 0
- static const uint16_t **FMD_MAX_MINUTIA_ANGLE** = 179
- static const uint16_t **FMD_MAX_MINUTIA_ISONC_ANGLE** = 255
- static const uint16_t **FMD_MAX_MINUTIA_ISOCC_ANGLE** = 63
- static const uint16_t **FMD_ANSI_ANGLE_UNIT** = 2
- static const uint16_t **FMD_ISO_ANGLE_UNIT**
- static const uint16_t **FMD_ISOCC_ANGLE_UNIT**
- static const uint16_t **FMD_MINUTIA_TYPE_OTHER** = 0
- static const uint16_t **FMD_MINUTIA_TYPE_RIDGE_ENDING** = 1
- static const uint16_t **FMD_MINUTIA_TYPE_BIFURCATION** = 2
- static const uint16_t **FMR_MIN_FINGER_QUALITY** = 0
- static const uint16_t **FMR_MAX_FINGER_QUALITY** = 100
- static const uint16_t **ISO_UNKNOWN_FINGER_QUALITY** = 0
- static const uint16_t **FED_RESERVED** = 0x0000
- static const uint16_t **FED_RIDGE_COUNT** = 0x0001
- static const uint16_t **FED_CORE_AND_DELTA** = 0x0002

- static const uint16_t **RCE_NONSPECIFIC** = 0x00
- static const uint16_t **RCE_FOUR_NEIGHBOR** = 0x01
- static const uint16_t **RCE_EIGHT_NEIGHBOR** = 0x02
- static const uint16_t **CORE_TYPE_NONANGULAR** = 0x00
- static const uint16_t **CORE_TYPE_ANGULAR** = 0x01
- static const uint16_t **DELTA_TYPE_NONANGULAR** = 0x00
- static const uint16_t **DELTA_TYPE_ANGULAR** = 0x01

E.43.1 Detailed Description

A class to represent a set of minutiae in an ANSI/INCITS record.

The base INCTISMinutiae class is responsible for reading minutiae data points and extended data. Each minutiae point, ridge count item, core, and delta is represented in the native ANSI/INCITS format. Objects of this base class cannot be instantiated, but rather derived classes are used to represent minutiae data taken from the INCITS-derived record formats.

E.43.2 Constructor & Destructor Documentation

E.43.2.1 BiometricEvaluation::Feature::INCITSMinutiae::INCITSMinutiae (const MinutiaPointSet & *mps*, const RidgeCountItemSet & *rcis*, const CorePointSet & *cps*, const DeltaPointSet & *dps*)

Construct an INCITS [Minutiae](#) object from its components.

The buffer index must be set to the location in the buffer to start reading minutiae data points and extended data.

Parameters

in	<i>mps</i>	The set of minutiae points.
in	<i>rcis</i>	The set of ridge count items.
in	<i>cps</i>	The set of core points.
in	<i>dps</i>	The set of delta points.

E.43.3 Member Function Documentation

E.43.3.1 void BiometricEvaluation::Feature::INCITSMinutiae::setMinutiaPoints (const MinutiaPointSet & *mps*)

Mutator for the minutiae point set.

Parameters

in	<i>mps</i>	The minutiae points.
----	------------	----------------------

E.43.3.2 void BiometricEvaluation::Feature::INCITSMinutiae::setRidgeCountItems (const RidgeCountItemSet & *rcis*)

Mutator for the ridge count items.

Parameters

in	<i>rcis</i>	The set of ridge count items.
----	-------------	-------------------------------

E.43.3.3 void BiometricEvaluation::Feature::INCITSMinutiae::setCorePointSet (const CorePointSet & *cps*)

Mutator for the set of core points.

Parameters

in	<i>cps</i>	The set of core points.
----	------------	-------------------------

E.43.3.4 void BiometricEvaluation::Feature::INCITSMinutiae::setDeltaPointSet (const DeltaPointSet & *dps*)

Mutator for the set of delta points.

Parameters

in	<i>dps</i>	The set of delta point items.
----	------------	-------------------------------

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

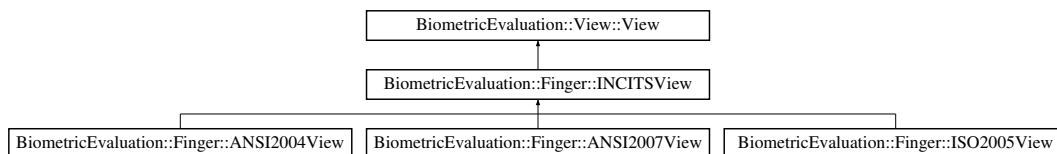
- be_feature_incitsminutiae.h

E.44 BiometricEvaluation::Finger::INCITSView Class Reference

A class to represent single finger view and derived information.

```
#include <be_finger_incitsview.h>
```

Inheritance diagram for BiometricEvaluation::Finger::INCITSView:

**Public Member Functions**

- [Feature::INCITSMinutiae getMinutiaeData \(\)](#) const
Obtain the set of minutiae records.
- [Finger::Position::Kind getPosition \(\)](#) const
Obtain the finger position.
- [Finger::Impression::Kind getImpressionType \(\)](#) const
Obtain the finger impression code.

- uint32_t [getQuality](#) () const
Obtain the finger quality value.
- uint16_t [getCaptureEquipmentID](#) () const
Obtain the capture equipment identifier.
- bool [isAppendixFCompliant](#) () const
Obtain the capture equipment compliance indicator for 'Appendix F'.
- tr1::shared_ptr< [Image::Image](#) > [getImage](#) () const
Obtain the image used for the finger view.
- [Image::Size](#) [getImageSize](#) () const
Obtain the image size.
- [Image::Resolution](#) [getImageResolution](#) () const
Obtain the image resolution.
- uint32_t [getImageDepth](#) () const
Obtain the image depth.
- [Image::CompressionAlgorithm::Kind](#) [getCompressionAlgorithm](#) () const
Obtain the compression algorithm used on the image.
- [Image::Resolution](#) [getScanResolution](#) () const
Obtain the image scan resolution.

Static Public Member Functions

- static [Finger::Position::Kind](#) [convertPosition](#) (int incitsFGP) throw (Error::DataError)
Convert a finger position code from an INCITS finger record to the common code.
- static [Finger::Impression::Kind](#) [convertImpression](#) (int incitsIMP) throw (Error::DataError)
Convert a impression type code from an INCITS finger record to the common code.

Static Public Attributes

- static const uint32_t **FMR_ANSI2004_STANDARD** = 1
- static const uint32_t **FMR_ISO2005_STANDARD** = 2
- static const uint32_t **FMR_ANSI2007_STANDARD** = 3
- static const string **FMR_BASE_FORMAT_ID**
- static const uint32_t **FMR_SPEC_VERSION_LEN** = 4
- static const string **FMR_BASE_SPEC_VERSION**
- static const string **FMR_ANSI2007_SPEC_VERSION**
- static const uint16_t **FMR_HDR_SCANNER_ID_MASK** = 0x0FFF
- static const uint16_t **FMR_HDR_COMPLIANCE_MASK** = 0xF000
- static const uint8_t **FMR_HDR_COMPLIANCE_SHIFT** = 12
- static const uint16_t **FMR_HDR_APPENDIX_F_MASK** = 0x0008
- static const uint8_t **FVMR_VIEW_NUMBER_MASK** = 0xF0
- static const uint8_t **FVMR_VIEW_NUMBER_SHIFT** = 4
- static const uint8_t **FVMR_IMPRESSION_MASK** = 0x0F

Protected Member Functions

- [INCITSView](#) (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber) throw (Error::DataError, Error::FileError)
Construct the common components of an INCITS finger view from records contained in files.
- [INCITSView](#) (const [Memory::uint8Array](#) &fmrBuffer, const [Memory::uint8Array](#) &firBuffer, const uint32_t viewNumber) throw (Error::DataError)
Construct an INCITS finger view from records contained in buffers.
- [Memory::uint8Array](#) const & [getFMRData](#) () const
Obtain a reference to the finger minutiae record data buffer.
- [Memory::uint8Array](#) const & [getFIRData](#) () const
Obtain a reference to the finger image record data buffer.
- void [setMinutiaeData](#) (const [Feature::INCITSMinutiae](#) &fmd)
Mutator for the [Feature::INCITSMinutiae](#) item.
- void [setPosition](#) (const Finger::Position::Kind &position)
Mutator for the position.
- void [setImpressionType](#) (const Finger::Impression::Kind &impression)
Mutator for the impression type.
- void [setQuality](#) (uint32_t quality)
Mutator for the finger quality value.
- void [setViewNumber](#) (uint32_t viewNumber)
Mutator for the finger view number.
- void [setCaptureEquipmentID](#) (uint16_t id)
Mutator for the equipment ID.
- void [setCBEFFProductIDs](#) (uint16_t owner, uint16_t type)
Mutator for the CBEFF Product ID owner and type.
- void [setAppendixFCompliance](#) (bool flag)
Mutator for the Appendix F compliance indicator.
- void [setImageSize](#) (const [Image::Size](#) &imageSize)
Mutator for the image size.
- void [setImageResolution](#) (const [Image::Resolution](#) &imageResolution)
Mutator for the image resolution.
- void [setScanResolution](#) (const [Image::Resolution](#) &scanResolution)
Mutator for the image scan resolution.
- void [setImageData](#) (const [Memory::uint8Array](#) &imageData)
Mutator for the image data.
- void [readFMRHeader](#) ([Memory::IndexedBuffer](#) &buf, const uint32_t formatStandard) throw (Error::ParameterError, Error::DataError)
Read the common finger minutiae record header from an INCITS record.
- void [readFVMR](#) ([Memory::IndexedBuffer](#) &buf) throw (Error::DataError)
Read the common finger view record information from an INCITS record.
- virtual [Feature::MinutiaPointSet](#) [readMinutiaeDataPoints](#) ([Memory::IndexedBuffer](#) &buf, uint32_t count) throw (Error::DataError)
Read the minutiae data points, and extended data blocks.
- virtual void [readExtendedDataBlock](#) ([Memory::IndexedBuffer](#) &buf) throw (Error::DataError)
Read the common extended data block.

- virtual Feature::RidgeCountItemSet [readRidgeCountData](#) (Memory::IndexedBuffer &buf, uint32_t dataLength) throw (Error::DataError)

Read the ridge count data.

- virtual void [readCoreDeltaData](#) (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)=0 throw (Error::DataError)

Read the core points data.

E.44.1 Detailed Description

A class to represent single finger view and derived information.

A base [Finger::INCITSView](#) object represents an INCITS/ANSI or ISO finger view. This class defines the common interface for all ANSI/ISO views as well as common implementations. Subclasses specialize this class in order to represent other versions of the ANSI/ISO specs. Objects of this class cannot be created.

E.44.2 Constructor & Destructor Documentation

E.44.2.1 [BiometricEvaluation::Finger::INCITSView::INCITSView](#) (const std::string & *fmrFilename*, const std::string & *firFilename*, const uint32_t *viewNumber*) throw (Error::DataError, Error::FileError)
[protected]

Construct the common components of an INCITS finger view from records contained in files.

See documentation in child classes of INCITS for information on constructing INCITS-derived finger views.

Parameters

in	<i>fmrFilename</i>	The name of the file containing the complete finger minutiae record.
in	<i>firFilename</i>	The name of the file containing the complete finger image record.
in	<i>viewNumber</i>	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.
Error::FileError	Could not open or read from file.

E.44.2.2 [BiometricEvaluation::Finger::INCITSView::INCITSView](#) (const Memory::uint8Array & *fmrBuffer*, const Memory::uint8Array & *firBuffer*, const uint32_t *viewNumber*) throw (Error::DataError)
[protected]

Construct an INCITS finger view from records contained in buffers.

See documentation in child classes of INCITS for information on constructing INCITS-derived finger views.

Parameters

in	<i>fmrBuffer</i>	The buffer containing the complete finger minutiae record.
in	<i>firBuffer</i>	The buffer containing the complete finger image record.
in	<i>viewNumber</i>	The finger view number to use.

Exceptions

<i>Error::DataError</i>	Invalid record format.
---	------------------------

E.44.3 Member Function Documentation

E.44.3.1 `static Finger::Position::Kind BiometricEvaluation::Finger::INCITSView::convertPosition (int incitsFGP) throw (Error::DataError) [static]`

Convert a finger position code from an INCITS finger record to the common code.

Parameters

in	<i>incitsFGP</i>	A finger position code as defined by the INCITS standard.
----	------------------	---

Exceptions

<i>Error::DataError</i>	The position code is invalid.
---	-------------------------------

Returns

The finger position code in common notation.

E.44.3.2 `static Finger::Impression::Kind BiometricEvaluation::Finger::INCITSView::convertImpression (int incitsIMP) throw (Error::DataError) [static]`

Convert a impression type code from an INCITS finger record to the common code.

Parameters

in	<i>incitsIMP</i>	A finger impression type code as defined by the INCITS standard.
----	------------------	--

Exceptions

<i>Error::DataError</i>	The impression type code is invalid.
---	--------------------------------------

Returns

The finger impression type code in common notation.

E.44.3.3 `Finger::Position::Kind BiometricEvaluation::Finger::INCITSView::getPosition () const`

Obtain the finger position.

Returns

The finger position.

E.44.3.4 `Finger::Impression::Kind BiometricEvaluation::Finger::INCITSView::getImpressionType () const`

Obtain the finger impression code.

Returns

The finger impression code.

E.44.3.5 `uint32_t BiometricEvaluation::Finger::INCITSView::getQuality () const`

Obtain the finger quality value.

Returns

The finger quality value.

E.44.3.6 `uint16_t BiometricEvaluation::Finger::INCITSView::getCaptureEquipmentID () const`

Obtain the capture equipment identifier.

Returns

The equipment ID.

E.44.3.7 `bool BiometricEvaluation::Finger::INCITSView::isAppendixFCompliant () const`

Obtain the capture equipment compliance indicator for 'Appendix F'.

Returns

True if 'Appendix F' compliant, false otherwise.

E.44.3.8 `tr1::shared_ptr<Image::Image> BiometricEvaluation::Finger::INCITSView::getImage () const`
[virtual]

Obtain the image used for the finger view.

Not all finger views will have an image, however the derived information, such as minutiae, may be present.

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

E.44.3.9 `Image::Size BiometricEvaluation::Finger::INCITSView::getImageSize () const` [virtual]

Obtain the image size.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image size must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

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

E.44.3.10 `Image::Resolution BiometricEvaluation::Finger::INCITSView::getImageResolution () const`
`[virtual]`

Obtain the image resolution.

[Image](#) resolution is taken from the biometric record, and not from the image data. In some cases, the resolution may be the components of the pixel ratio, and applications must check the [Image::Resolution::units](#) field for value NA.

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

E.44.3.11 `uint32_t BiometricEvaluation::Finger::INCITSView::getImageDepth () const` `[virtual]`

Obtain the image depth.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image depth must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

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

E.44.3.12 `Image::CompressionAlgorithm::Kind BiometricEvaluation::Finger::INCITSView::getCompression-
Algorithm () const` `[virtual]`

Obtain the compression algorithm used on the image.

This value is as present in the biometric record, and not obtained from the image data itself.

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

E.44.3.13 `Image::Resolution BiometricEvaluation::Finger::INCITSView::getScanResolution () const`
`[virtual]`

Obtain the image scan resolution.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image resolution must be equal, but applications can check for inconsistencies.

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

E.44.3.14 `Memory::uint8Array const& BiometricEvaluation::Finger::INCITSView::getFMRData () const`
`[protected]`

Obtain a reference to the finger minutiae record data buffer.

Returns

The entire finger minutiae record data.

E.44.3.15 `Memory::uint8Array const& BiometricEvaluation::Finger::INCITSView::getFIRData () const`
`[protected]`

Obtain a reference to the finger image record data buffer.

Returns

The entire finger image record data.

E.44.3.16 `void BiometricEvaluation::Finger::INCITSView::setMinutiaeData (const Feature::INCITSMinutiae & fmd)` [protected]

Mutator for the [Feature::INCITSMinutiae](#) item.

Parameters

in	<i>fmd</i>	The minutiae data object.
----	------------	---------------------------

E.44.3.17 `void BiometricEvaluation::Finger::INCITSView::setPosition (const Finger::Position::Kind & position)` [protected]

Mutator for the position.

Parameters

in	<i>position</i>	The finger position.
----	-----------------	----------------------

E.44.3.18 `void BiometricEvaluation::Finger::INCITSView::setImpressionType (const Finger::Impression::Kind & impression)` [protected]

Mutator for the impression type.

Parameters

in	<i>impression</i>	The finger impression type code.
----	-------------------	----------------------------------

E.44.3.19 `void BiometricEvaluation::Finger::INCITSView::setQuality (uint32_t quality)` [protected]

Mutator for the finger quality value.

Parameters

in	<i>quality</i>	The quality value.
----	----------------	--------------------

E.44.3.20 `void BiometricEvaluation::Finger::INCITSView::setViewNumber (uint32_t viewNumber)` [protected]

Mutator for the finger view number.

Parameters

in	<i>viewNumber</i>	The view number value.
----	-------------------	------------------------

E.44.3.21 void BiometricEvaluation::Finger::INCITSView::setCaptureEquipmentID (uint16_t *id*)
[protected]

Mutator for the equipment ID.

Parameters

in	<i>id</i>	The equipment ID value.
----	-----------	-------------------------

E.44.3.22 void BiometricEvaluation::Finger::INCITSView::setCBEFFProductIDs (uint16_t *owner*, uint16_t *type*) [protected]

Mutator for the CBEFF Product ID owner and type.

Parameters

in	<i>owner</i>	The CBEFF ID of the product owner.
in	<i>type</i>	The CBEFF ID of the product type.

E.44.3.23 void BiometricEvaluation::Finger::INCITSView::setAppendixFCompliance (bool *flag*)
[protected]

Mutator for the Appendix F compliance indicator.

Parameters

in	<i>flag</i>	True if the capture equipment is 'Appendix F' compliant, false if not.
----	-------------	--

E.44.3.24 void BiometricEvaluation::Finger::INCITSView::setImageSize (const Image::Size & *imageSize*)
[protected]

Mutator for the image size.

Parameters

in	<i>imageSize</i>	The image size object.
----	------------------	------------------------

E.44.3.25 void BiometricEvaluation::Finger::INCITSView::setImageResolution (const Image::Resolution & *imageResolution*) [protected]

Mutator for the image resolution.

Parameters

in	<i>image-Resolution</i>	The image resolution object.
----	-------------------------	------------------------------

E.44.3.26 void BiometricEvaluation::Finger::INCITSView::setScanResolution (const Image::Resolution & *scanResolution*) [protected]

Mutator for the image scan resolution.

Parameters

in	<i>scanResolution</i>	The image scan resolution object.
----	-----------------------	-----------------------------------

E.44.3.27 void BiometricEvaluation::Finger::INCITSView::setImageData (const Memory::uint8Array & *imageData*) [protected]

Mutator for the image data.

Parameters

in	<i>imageData</i>	The image data object.
----	------------------	------------------------

E.44.3.28 void BiometricEvaluation::Finger::INCITSView::readFMRHeader (Memory::IndexedBuffer & *buf*, const uint32_t *formatStandard*) throw (Error::ParameterError, Error::DataError) [protected]

Read the common finger minutiae record header from an INCITS record.

For ANSI-2004 and ISO-2005 record formats, the finger minutiae record header is (mostly) the same.

Parameters

in	<i>buf</i>	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the header.
in	<i>formatStandard</i>	Value indicating which header version to read; one of FMR_ANSI2004_STANDARD or FMR_ISO2005_STANDARD.

Exceptions

<i>ParameterError</i>	The specVersion parameter is incorrect.
<i>DataError</i>	The INCITS record has invalid or missing data.

Reimplemented in [BiometricEvaluation::Finger::ANSI2007View](#).

E.44.3.29 `void BiometricEvaluation::Finger::INCITSView::readFVMR (Memory::IndexedBuffer & buf) throw (Error::DataError) [protected]`

Read the common finger view record information from an INCITS record.

A [Finger View](#) from an INCITS record includes image information, minutiae, and extended data ridge counts, cores/deltas, etc.) For ANSI-2004 and ISO-2005 record formats, the finger view representation is the same, so this functions parses those record formats. The minutiae data items are also read, as well as any extended data.

Parameters

in, out	<i>buf</i>	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the finger view, including the extended data.
---------	------------	--

Exceptions

<i>DataError</i>	The INCITS record has invalid or missing data.
------------------	--

Reimplemented in [BiometricEvaluation::Finger::ANSI2007View](#).

E.44.3.30 `virtual Feature::MinutiaPointSet BiometricEvaluation::Finger::INCITSView::readMinutiaeDataPoints (Memory::IndexedBuffer & buf, uint32_t count) throw (Error::DataError) [protected, virtual]`

Read the minutiae data points, and extended data blocks.

Function to be implemented by derived classes to read the minutiae data points and extended data block according to the specific standard they represent.

Parameters

in	<i>buf</i>	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the finger view, including the extended data.
in	<i>count</i>	Number of minutiae data points to read.

Exceptions

<i>DataError</i>	The INCITS record has invalid or missing data.
------------------	--

E.44.3.31 `virtual void BiometricEvaluation::Finger::INCITSView::readExtendedDataBlock (Memory::IndexedBuffer & buf) throw (Error::DataError) [protected, virtual]`

Read the common extended data block.

Parameters

in, out	<i>buf</i>	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the extended data block.
---------	------------	---

Exceptions

<i>DataError</i>	The INCITS record has invalid or missing data.
------------------	--

E.44.3.32 `virtual Feature::RidgeCountItemSet BiometricEvaluation::Finger::INCITSView::readRidgeCountData (Memory::IndexedBuffer & buf, uint32_t dataLength) throw (Error::DataError) [protected, virtual]`

Read the ridge count data.

This method reads data in the base INCITS format as defined in INCITS/ANSI 378-2004. This method may be overridden by derived classes to read data in a different record format.

Parameters

in, out	<i>buf</i>	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last ridge count item.
in	<i>dataLength</i>	The length of the entire ridge count data block.

E.44.3.33 `virtual void BiometricEvaluation::Finger::INCITSView::readCoreDeltaData (Memory::IndexedBuffer & buf, uint32_t dataLength, Feature::CorePointSet & cores, Feature::DeltaPointSet & deltas) throw (Error::DataError) [protected, pure virtual]`

Read the core points data.

This method must be overridden by derived classes to read data in a specific record format.

Parameters

in, out	<i>buf</i>	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last core point data item.
out	<i>cores</i>	The set of core data items.
out	<i>deltas</i>	The set of delta data items.
in	<i>dataLength</i>	The length of the entire ridge count data block.

Implemented in [BiometricEvaluation::Finger::ANSI2007View](#), [BiometricEvaluation::Finger::ISO2005View](#), and [BiometricEvaluation::Finger::ANSI2004View](#).

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

- `be_finger_incitsview.h`

E.45 BiometricEvaluation::Memory::IndexedBuffer Class Reference

Manage a memory buffer with an index.

```
#include <be_memory_indexedbuffer.h>
```

Public Member Functions

- **operator uint8_t * ()**
- **uint8_t * operator-> ()**
- **IndexedBuffer & operator=** (const [IndexedBuffer](#) &other)
- **IndexedBuffer ()**
Create an indexed buffer of zero length.
- **IndexedBuffer** (uint32_t size)
Create an indexed buffer of a given length.
- **IndexedBuffer** (uint8_t *data, uint32_t size)
Create an indexed buffer around an existing buffer of a given length.
- **IndexedBuffer** (const [IndexedBuffer](#) ©)
Copy constructor.
- **uint32_t getSize ()**
Obtain the current size of the buffer.
- **uint32_t getIndex ()**
Obtain the current index into the buffer.
- **void setIndex** (uint32_t index) throw (Error::ParameterError)
Set the current index into the buffer.
- **uint8_t scanU8Val ()** throw (Error::DataError)
Obtain the next element of the buffer and increment the current index value.
- **uint16_t scanU16Val ()** throw (Error::DataError)
Obtain the next two elements of the buffer and increment the current index value.
- **uint16_t scanBeU16Val ()** throw (Error::DataError)
Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.
- **uint32_t scanU32Val ()** throw (Error::DataError)
Obtain the next four elements of the buffer and increment the current index value by four.
- **uint32_t scanBeU32Val ()** throw (Error::DataError)
Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.
- **uint64_t scanU64Val ()** throw (Error::DataError)
Obtain the next eight elements of the buffer and increment the current index value by eight.
- **uint32_t scan** (void *buf, const uint32_t len) throw (Error::DataError)
Obtain the next 'n' elements of the buffer and increment the current index value by n.
- **uint8_t & operator[]** (ptrdiff_t i)
Subscripting operator.
- **const uint8_t & operator[]** (ptrdiff_t i) const
Constant subscripting operator.

E.45.1 Detailed Description

Manage a memory buffer with an index.

The memory buffer is treated as an array of unsigned eight bit values. This class provides safe access to the array with methods to retrieve 8/16/32/64-bit elements, or an arbitrary segment starting at the index, from the array while advancing the current index. An exception is thrown by these methods whenever the retrieval would reach beyond the size of the buffer.

The buffer can also be accessed directly by subscripting.

E.45.2 Constructor & Destructor Documentation

E.45.2.1 BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (uint8_t * *data*, uint32_t *size*)

Create an indexed buffer around an existing buffer of a given length.

An object constructed in this manner will not free the underlying data buffer.

E.45.3 Member Function Documentation

E.45.3.1 uint32_t BiometricEvaluation::Memory::IndexedBuffer::getSize ()

Obtain the current size of the buffer.

Returns

The current buffer size.

E.45.3.2 uint32_t BiometricEvaluation::Memory::IndexedBuffer::getIndex ()

Obtain the current index into the buffer.

Returns

The current buffer index.

E.45.3.3 void BiometricEvaluation::Memory::IndexedBuffer::setIndex (uint32_t *index*) throw (Error::ParameterError)

Set the current index into the buffer.

Parameters

<i>in</i>	<i>index</i>	The index value to set.
-----------	--------------	-------------------------

Exceptions

<i>Error::ParameterError</i>	The index parameter is too large.
--	-----------------------------------

E.45.3.4 uint8_t BiometricEvaluation::Memory::IndexedBuffer::scanU8Val () throw (Error::DataError)

Obtain the next element of the buffer and increment the current index value.

Exceptions

<i>Error::DataError</i>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 8-bit value.

E.45.3.5 `uint16_t BiometricEvaluation::Memory::IndexedBuffer::scanU16Val () throw (Error::DataError)`

Obtain the next two elements of the buffer and increment the current index value.

Exceptions

<i>Error::DataError</i>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 16-bit value.

E.45.3.6 `uint16_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU16Val () throw (Error::DataError)`

Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.

Exceptions

<i>Error::DataError</i>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 16-bit value.

E.45.3.7 `uint32_t BiometricEvaluation::Memory::IndexedBuffer::scanU32Val () throw (Error::DataError)`

Obtain the next four elements of the buffer and increment the current index value by four.

Exceptions

<i>Error::DataError</i>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 32-bit value.

E.45.3.8 `uint32_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU32Val () throw (Error::DataError)`

Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.

Exceptions

<i>Error::DataError</i>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 32-bit value.

E.45.3.9 uint64_t BiometricEvaluation::Memory::IndexedBuffer::scanU64Val () throw (Error::DataError)

Obtain the next eight elements of the buffer and increment the current index value by eight.

Exceptions

<i>Error::DataError</i>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 64-bit value.

E.45.3.10 uint32_t BiometricEvaluation::Memory::IndexedBuffer::scan (void * buf, const uint32_t len) throw (Error::DataError)

Obtain the next 'n' elements of the buffer and increment the current index value by n.

Parameters

in	<i>buf</i>	Buffer to store the copied data. Can be NULL. The current index is incremented.
in	<i>len</i>	The number of elements to copy.

Exceptions

<i>Error::DataError</i>	The buffer is exhausted.
---	--------------------------

Returns

The number of elements copied.

E.45.3.11 uint8_t& BiometricEvaluation::Memory::IndexedBuffer::operator[] (ptrdiff_t i)

Subscripting operator.

Provides array-like access to elements of the buffer. This operation will not affect the current index value.

Parameters

in	<i>i</i>	The subscript.
----	----------	----------------

Returns

Reference to element 'i' of the buffer.

E.45.3.12 `const uint8_t& BiometricEvaluation::Memory::IndexedBuffer::operator[] (ptrdiff_t i) const`

Constant subscripting operator.

Provides read-only array-like access to elements of the buffer. This operation will not affect the current index value.

Parameters

<code>in</code>	<code>i</code>	The subscript.
-----------------	----------------	----------------

Returns

Reference to const element 'i' of the buffer.

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

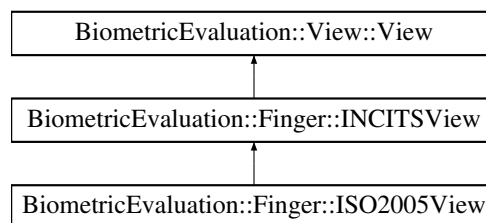
- `be_memory_indexedbuffer.h`

E.46 **BiometricEvaluation::Finger::ISO2005View Class Reference**

A class to represent single finger view and derived information.

```
#include <be_finger_iso2005view.h>
```

Inheritance diagram for BiometricEvaluation::Finger::ISO2005View:

**Public Member Functions**

- [ISO2005View](#) (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view-Number) throw (Error::DataError, Error::FileError)

Construct an ISO-2005 finger view from records contained in files.

- [ISO2005View](#) ([Memory::uint8Array](#) &fmrBuffer, [Memory::uint8Array](#) &firBuffer, const uint32_t view-Number) throw (Error::DataError)

Construct an ISO-2005 finger view from records contained in buffers.

Static Public Attributes

- static const uint16_t **CORE_TYPE_MASK** = 0xC000
- static const uint16_t **CORE_TYPE_SHIFT** = 14
- static const uint16_t **CORE_NUM_CORES_MASK** = 0x3F
- static const uint16_t **CORE_X_COORD_MASK** = 0x3FFF
- static const uint16_t **CORE_Y_COORD_MASK** = 0x3FFF
- static const uint16_t **CORE_MIN_NUM** = 0
- static const uint16_t **DELTA_TYPE_MASK** = 0xC000
- static const uint16_t **DELTA_TYPE_SHIFT** = 14
- static const uint16_t **DELTA_NUM_DELTAS_MASK** = 0x3F
- static const uint16_t **DELTA_X_COORD_MASK** = 0x3FFF
- static const uint16_t **DELTA_Y_COORD_MASK** = 0x3FFF

Protected Member Functions

- virtual void [readCoreDeltaData](#) ([Memory::IndexedBuffer](#) &buf, uint32_t dataLength, [Feature::CorePointSet](#) &cores, [Feature::DeltaPointSet](#) &deltas) throw ([Error::DataError](#))

Read the core points data.

E.46.1 Detailed Description

A class to represent single finger view and derived information.

A [Finger::ISO2005View](#) object represents a finger view from a ISO/IEC-2005 [Finger](#) Minutiae Record.

E.46.2 Constructor & Destructor Documentation

- E.46.2.1** [BiometricEvaluation::Finger::ISO2005View::ISO2005View](#) (const std::string & *fmrFilename*, const std::string & *firFilename*, const uint32_t *viewNumber*) throw ([Error::DataError](#), [Error::FileError](#))

Construct an ISO-2005 finger view from records contained in files.

A view can be constructed from a single record, with information missing as appropriate. For example, if a view is constructed with just the minutiae record, no image would be part of the view. However, the image size etc. would be present because that information is also present in the minutiae record.

Parameters

in	<i>fmrFilename</i>	The name of the file containing the complete finger minutiae record.
in	<i>firFilename</i>	The name of the file containing the complete finger image record.
in	<i>viewNumber</i>	The finger view number to use.

- E.46.2.2** [BiometricEvaluation::Finger::ISO2005View::ISO2005View](#) ([Memory::uint8Array](#) & *fmrBuffer*, [Memory::uint8Array](#) & *firBuffer*, const uint32_t *viewNumber*) throw ([Error::DataError](#))

Construct an ISO-2005 finger view from records contained in buffers.

A view can be constructed from a single record, with information missing as appropriate. For example, if a

view is constructed with just the minutiae record, no image would be part of the view. However, the image size etc. would be present because that information is also present in the minutiae record.

Parameters

in	<i>fmrBuffer</i>	The buffer containing the complete finger minutiae record.
in	<i>firBuffer</i>	The buffer containing the complete finger image record.
in	<i>viewNumber</i>	The finger view number to use.

Exceptions

<i>Error::DataError</i>	Invalid record format.
---	------------------------

E.46.3 Member Function Documentation

E.46.3.1 `virtual void BiometricEvaluation::Finger::ISO2005View::readCoreDeltaData (Memory::IndexedBuffer & buf, uint32_t dataLength, Feature::CorePointSet & cores, Feature::DeltaPointSet & deltas) throw (Error::DataError) [protected, virtual]`

Read the core points data.

This method must be overridden by derived classes to read data in a specific record format.

Parameters

in, out	<i>buf</i>	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last core point data item.
out	<i>cores</i>	The set of core data items.
out	<i>deltas</i>	The set of delta data items.
in	<i>dataLength</i>	The length of the entire ridge count data block.

Implements [BiometricEvaluation::Finger::INCITSView](#).

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

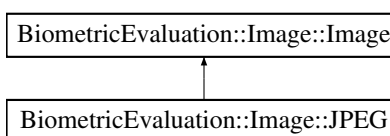
- `be_finger_iso2005view.h`

E.47 BiometricEvaluation::Image::JPEG Class Reference

A JPEG-encoded image.

```
#include <be_image_jpeg.h>
```

Inheritance diagram for BiometricEvaluation::Image::JPEG:



Public Member Functions

- **JPEG** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- **Memory::AutoArray**< uint8_t > **getRawGrayscaleData** (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)
Accessor for decompressed data in grayscale.
- **Memory::AutoArray**< uint8_t > **getRawData** () const throw (Error::DataError)
Accessor for the raw image data. The data returned should not be compressed or encoded.

Static Public Member Functions

- static bool **isJPEG** (const uint8_t *data, const size_t size)
- static int **getc_skip_marker_segment** (const unsigned short marker, unsigned char **bufptr, unsigned char *ebufptr)

E.47.1 Detailed Description

A JPEG-encoded image.

E.47.2 Member Function Documentation

E.47.2.1 **Memory::AutoArray**<uint8_t> **BiometricEvaluation::Image::JPEG::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

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

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements **BiometricEvaluation::Image::Image**.

E.47.2.2 `Memory::AutoArray<uint8_t> BiometricEvaluation::Image::JPEG::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

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

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

E.47.2.3 `static bool BiometricEvaluation::Image::JPEG::isJPEG (const uint8_t * data, const size_t size) [static]`

Whether or not data is a Lossy [JPEG](#) image.

Parameters

in	<i>data</i>	The buffer to check.
in	<i>size</i>	The size of data.

Returns

true if data appears to be a Lossy [JPEG](#) image, false otherwise

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

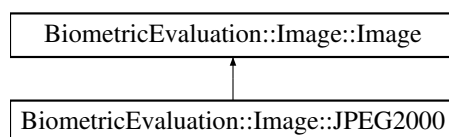
- `be_image_jpeg.h`

E.48 BiometricEvaluation::Image::JPEG2000 Class Reference

A JPEG-2000-encoded image.

```
#include <be_image_jpeg2000.h>
```

Inheritance diagram for `BiometricEvaluation::Image::JPEG2000`:



Public Member Functions

- [JPEG2000](#) (const uint8_t *data, const uint64_t size, const int8_t codec=2) throw (Error::DataError, Error::StrategyError)
Create a new [JPEG2000](#) object.
- [Memory::AutoArray](#)< uint8_t > [getRawData](#) () const throw (Error::DataError)
Accessor for the raw image data. The data returned should not be compressed or encoded.
- [Memory::AutoArray](#)< uint8_t > [getRawGrayscaleData](#) (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)
Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool [isJPEG2000](#) (const uint8_t *data)

E.48.1 Detailed Description

A JPEG-2000-encoded image.

E.48.2 Constructor & Destructor Documentation

E.48.2.1 [BiometricEvaluation::Image::JPEG2000::JPEG2000](#) (const uint8_t * *data*, const uint64_t *size*, const int8_t *codec* = 2) throw (Error::DataError, Error::StrategyError)

Create a new [JPEG2000](#) object.

Parameters

in	<i>data</i>	The image data.
in	<i>size</i>	The size of the image data, in bytes.
in	<i>codec</i>	The codec used to encode data.

Exceptions

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

E.48.3 Member Function Documentation

E.48.3.1 [Memory::AutoArray](#)<uint8_t> [BiometricEvaluation::Image::JPEG2000::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

<i>Error::DataError</i>	Error decompressing image data.
---	---

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

E.48.3.2 `Memory::AutoArray<uint8_t> BiometricEvaluation::Image::JPEG2000::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

<i>Error::DataError</i>	Error decompressing image data.
<i>Error::ParameterError</i>	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](#).

E.48.3.3 `static bool BiometricEvaluation::Image::JPEG2000::isJPEG2000 (const uint8_t * data) [static]`

Whether or not *data* is a JPEG-2000 image.

Parameters

<i>in</i>	<i>data</i>	The buffer to check.
-----------	-------------	----------------------

Returns

true if *data* appears to be a JPEG-2000 image, false otherwise.

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

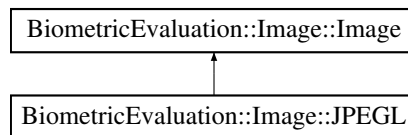
- `be_image_jpeg2000.h`

E.49 BiometricEvaluation::Image::JPEGL Class Reference

A Lossless JPEG-encoded image.

```
#include <be_image_jpeg1.h>
```

Inheritance diagram for BiometricEvaluation::Image::JPEGL:



Public Member Functions

- **JPEGL** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- [Memory::AutoArray](#)< uint8_t > [getRawGrayscaleData](#) (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)
Accessor for decompressed data in grayscale.
- [Memory::AutoArray](#)< uint8_t > [getRawData](#) () const throw (Error::DataError)
Accessor for the raw image data. The data returned should not be compressed or encoded.

Static Public Member Functions

- static bool [isJPEGL](#) (const uint8_t *data, const size_t size)

E.49.1 Detailed Description

A Lossless JPEG-encoded image.

E.49.2 Member Function Documentation

E.49.2.1 [Memory::AutoArray](#)<uint8_t> [BiometricEvaluation::Image::JPEGL::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

<i>Error::DataError</i>	Error decompressing image data.
<i>Error::ParameterError</i>	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](#).

E.49.2.2 `Memory::AutoArray<uint8_t> BiometricEvaluation::Image::JPEG::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

<i>Error::DataError</i>	Error decompressing image data.
---	---------------------------------

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

E.49.2.3 `static bool BiometricEvaluation::Image::JPEG::isJPEG (const uint8_t * data, const size_t size) [static]`

Whether or not data is a Lossless [JPEG](#) image.

Parameters

in	<i>data</i>	The buffer to check.
in	<i>size</i>	The size of data.

Returns

true if data appears to be a Lossless [JPEG](#) image, false otherwise.

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

- `be_image_jpeg.h`

E.50 BiometricEvaluation::IO::LogCabinet Class Reference

```
#include <be_io_logcabinet.h>
```

Public Member Functions

- [LogCabinet](#) (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- [LogCabinet](#) (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- `tr1::shared_ptr< LogSheet > newLogSheet` (const string &name, const string &description) throw (Error::ObjectExists, Error::StrategyError)
- string [getName](#) ()
- string [getDescription](#) ()
- unsigned int [getCount](#) ()

Static Public Member Functions

- static void [remove](#) (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)

E.50.1 Detailed Description

A class to represent a collection of log sheets.

E.50.2 Constructor & Destructor Documentation

E.50.2.1 BiometricEvaluation::IO::LogCabinet::LogCabinet (const string & *name*, const string & *description*, const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new [LogCabinet](#) in the file system.

Parameters

in	<i>name</i>	The name of the LogCabinet to be created.
in	<i>description</i>	The text used to describe the cabinet.
in	<i>parentDir</i>	Where, in the file system, the cabinet is to be stored. This directory must exist.

Returns

An object representing the new log cabinet.

Exceptions

Error::ObjectExists	The cabinet was previously created.
Error::StrategyError	
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

E.50.2.2 BiometricEvaluation::IO::LogCabinet::LogCabinet (const string & *name*, const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing [LogCabinet](#).

Parameters

in	<i>name</i>	The name of the LogCabinet to be created.
in	<i>parentDir</i>	Where, in the file system, the cabinet is to be stored. This directory must exist.

Returns

An object representing the log cabinet.

Exceptions

Error::ObjectDoesNotExist	The cabinet does not exist in the file system.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

E.50.3 Member Function Documentation

E.50.3.1 tr1::shared_ptr<LogSheet> BiometricEvaluation::IO::LogCabinet::newLogSheet (const string & *name*, const string & *description*) throw (Error::ObjectExists, Error::StrategyError)

Create a new [LogSheet](#) within the [LogCabinet](#).

Parameters

in	<i>name</i>	The name of the LogSheet to be created.
in	<i>description</i>	The text used to describe the sheet. This text is written into the log file prior to any entries.

Returns

An object pointer to the new log sheet.

Exceptions

Error::ObjectExists	The sheet was previously created.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

E.50.3.2 string BiometricEvaluation::IO::LogCabinet::getName ()

Obtain the name of the [LogCabinet](#).

@ returns The name of the [LogCabinet](#).

E.50.3.3 string BiometricEvaluation::IO::LogCabinet::getDescription ()

Obtain the description of the [LogCabinet](#).

@ returns The description of the [LogCabinet](#).

E.50.3.4 unsigned int BiometricEvaluation::IO::LogCabinet::getCount ()

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

@ returns The number of LogSheets manages by the cabinet.

E.50.3.5 static void BiometricEvaluation::IO::LogCabinet::remove (const string & name, const string & parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]

Remove a [LogCabinet](#).

Parameters

in	<i>name</i>	The name of the LogCabinet to be removed.
in	<i>parentDir</i>	Where, in the file system, the sheet is to be stored. This directory must exist.

Exceptions

<i>Error::ObjectDoesNotExist</i>	The LogCabinet does not exist.
<i>Error::StrategyError</i>	An error occurred when using the underlying file system, or name or parentDir is malformed.

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

- be_io_logcabinet.h

E.51 BiometricEvaluation::IO::LogSheet Class Reference

A class to represent a single logging mechanism.

```
#include <be_io_logsheets.h>
```

Public Member Functions

- [LogSheet](#) (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
Create a new log sheet.
- [LogSheet](#) (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)

- *Open an existing new log sheet for appending.*
- virtual `~LogSheet ()`
- virtual void `write (const string &entry) throw (Error::StrategyError)`
- *Write a string as an entry to the log file.*
- virtual void `writeComment (const string &comment) throw (Error::StrategyError)`
- *Write a string as a comment to the log file.*
- virtual void `newEntry () throw (Error::StrategyError)`
- *Start a new entry, causing the existing entry to be closed.*
- virtual string `getCurrentEntry ()`
- *Obtain the contents of the current entry currently under construction.*
- virtual void `resetCurrentEntry ()`
- virtual uint32_t `getCurrentEntryNumber ()`
- *Obtain the current entry number.*
- virtual void `sync () throw (Error::StrategyError)`
- *Synchronize any buffered data to the underlying log file.*
- void `setAutoSync (bool state)`
- string `sequence (bool comments=false, bool trim=true, int32_t cursor=BE_LOGSHEET_SEQ_NEXT) throw (Error::FileError, Error::ObjectDoesNotExist, Error::StrategyError)`
- *Sequence through a [LogSheet](#), returning one entry per invocation.*

Static Public Member Functions

- static string `trim (const string &entry)`
- *Trim delimiters from [LogSheet](#) entries.*
- static void `mergeLogSheets (vector< tr1::shared_ptr< LogSheet > > &logSheets) throw (Error::FileError, Error::StrategyError)`
- *Merge multiple [LogSheets](#) into a single [LogSheet](#).*

Static Public Attributes

- static const char `CommentDelimiter = '#'`
- static const char `EntryDelimiter = 'E'`
- static const string `DescriptionTag`
- static const int32_t `BE_LOGSHEET_SEQ_START = 1`
- static const int32_t `BE_LOGSHEET_SEQ_NEXT = 2`

Protected Member Functions

- `LogSheet (const LogSheet &)`
- `LogSheet & operator= (const LogSheet &)`
- void `updateCursor () throw (Error::FileError)`
- *Update the cursor position of the sequence file.*

Protected Attributes

- uint32_t [_entryNumber](#)
- auto_ptr< std::fstream > [_theLogFile](#)
- bool [_autoSync](#)
- tr1::shared_ptr< std::fstream > [_sequenceFile](#)
- streamoff [_cursor](#)

E.51.1 Detailed Description

A class to represent a single logging mechanism.

A [LogSheet](#) is a string stream, so applications can write into the stream as a staging area using the << operator, then start a new entry by calling [newEntry\(\)](#). Entries in the log file are prefixed with an entry number, which is incremented when the entry is written (either by directly calling [write\(\)](#), or calling [newEntry\(\)](#)).

A [LogSheet](#) object can be constructed and passed back to the client by the [LogCabinet](#) object. All sheets created in the manner are placed in a common area maintained by the cabinet.

Note

By default, the entries in the [LogSheet](#) may not be immediately written to the file system, depending on the buffering behavior of the operating system. Applications can force a write by invoking [sync\(\)](#), or force a write at every new log entry by invoking [setAutoSync\(true\)](#).

Entries created by applications may be composed of more than one line (each separated by the newline character). The text at the beginning of a line should not "look like" an entry number:

Edddd

i.e. the entry delimiter followed by some digits. [LogSheet](#) won't check for that condition, but any existing [LogSheet](#) that is re-opened for append may have an incorrect starting entry number.

E.51.2 Constructor & Destructor Documentation

E.51.2.1 [BiometricEvaluation::IO::LogSheet::LogSheet \(const string & name, const string & description, const string & parentDir \) throw \(Error::ObjectExists, Error::StrategyError\)](#)

Create a new log sheet.

Parameters

in	<i>name</i>	The name of the LogSheet to be created.
in	<i>description</i>	The text used to describe the sheet. This text is written into the log file prior to any entries.
in	<i>parentDir</i>	Where, in the file system, the sheet is to be stored. This directory must exist.

Returns

An object representing the new log sheet.

Exceptions

<i>Error::ObjectExists</i>	The sheet was previously created.
<i>Error::StrategyError</i>	An error occurred when using the underlying file system, or name or parentDir is malformed.

E.51.2.2 `BiometricEvaluation::IO::LogSheet::LogSheet (const string & name, const string & parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)`

Open an existing new log sheet for appending.

On open, the current entry counter is set to the last entry number plus one.

Note

Opening a large [LogSheet](#) may be a costly operation.

Parameters

in	<i>name</i>	The name of the LogSheet to be opened.
in	<i>parentDir</i>	Where, in the file system, the sheet is stored.

Returns

An object representing the existing log sheet.

Exceptions

<i>Error::ObjectDoesNotExist</i>	The sheet does not exist.
<i>Error::StrategyError</i>	An error occurred when using the underlying file system, or name or parentDir is malformed.

E.51.2.3 `virtual BiometricEvaluation::IO::LogSheet::~~LogSheet ()` [virtual]

Destructor

E.51.2.4 `BiometricEvaluation::IO::LogSheet::LogSheet (const LogSheet &)` [protected]

Prevent copying of [LogSheet](#) objects

E.51.3 Member Function Documentation

E.51.3.1 `virtual void BiometricEvaluation::IO::LogSheet::write (const string & entry) throw (Error::StrategyError)` [virtual]

Write a string as an entry to the log file.

This does not affect the current log entry buffer, but does increment the entry number.

Parameters

<i>in</i>	<i>entry</i>	The text of the log entry.
-----------	--------------	----------------------------

Exceptions

<i>Error::StrategyError</i>	An error occurred when using the underlying file system.
---	--

E.51.3.2 `virtual void BiometricEvaluation::IO::LogSheet::writeComment (const string & comment) throw (Error::StrategyError) [virtual]`

Write a string as a comment to the log file.

This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

<i>in</i>	<i>comment</i>	The text of the comment.
-----------	----------------	--------------------------

Exceptions

<i>Error::StrategyError</i>	An error occurred when using the underlying file system.
---	--

E.51.3.3 `virtual void BiometricEvaluation::IO::LogSheet::newEntry () throw (Error::StrategyError) [virtual]`

Start a new entry, causing the existing entry to be closed.

Applications do not have to call this method for the first entry, however, as the stream is ready for writing upon construction.

Exceptions

<i>Error::StrategyError</i>	An error occurred when using the underlying file system.
---	--

E.51.3.4 `virtual string BiometricEvaluation::IO::LogSheet::getCurrentEntry () [virtual]`

Obtain the contents of the current entry currently under construction.

Returns

The text of the current entry.

E.51.3.5 `virtual void BiometricEvaluation::IO::LogSheet::resetCurrentEntry () [virtual]`

Reset the current entry buffer to the beginning.

E.51.3.6 `virtual uint32_t BiometricEvaluation::IO::LogSheet::getCurrentEntryNumber () [virtual]`

Obtain the current entry number.

Returns

The current entry number.

E.51.3.7 `virtual void BiometricEvaluation::IO::LogSheet::sync () throw (Error::StrategyError) [virtual]`

Synchronize any buffered data to the underlying log file.

This syncing is dependent on the behavior of the underlying filesystem and operating system.

Exceptions

<i>Error::StrategyError</i>	An error occurred when using the underlying file system.
---	--

E.51.3.8 `void BiometricEvaluation::IO::LogSheet::setAutoSync (bool state)`

Turn on/off auto-sync of the data. Applications can gain login performance by turning off auto-sync, or gain reliability by turning it on.

Parameters

<i>state</i>	When true, the data is sync'd whenever newEntry() is or write() is called. When false, sync() must be called to force a write.
--------------	--

E.51.3.9 `string BiometricEvaluation::IO::LogSheet::sequence (bool comments = false, bool trim = true, int32_t cursor = BE_LOGSHEET_SEQ_NEXT) throw (Error::FileError, Error::ObjectDoesNotExist, Error::StrategyError)`

Sequence through a [LogSheet](#), returning one entry per invocation.

Parameters

<i>comments</i>	Include comments when sequencing
<i>trim</i>	Whether or not to include entry delimiters.
<i>cursor</i>	The location within the sequence to return.

Returns

The contents of the sequenced entry, as was originally given to [write\(\)](#).

Exceptions

Error::FileError , Error	occured while performing file IO .
Error::ObjectDoesNotExist	The LogSheet cannot be found on disk.
Error::StrategyError	Invalid cursor position or the contents of the LogSheet is malformed.

E.51.3.10 static string BiometricEvaluation::IO::LogSheet::trim (const string & entry) [static]

Trim delimiters from [LogSheet](#) entries.

Works for comments and numbered entries.

Parameters

in	entry	The entry to trim.
----	-------	--------------------

Returns

Delimiter-less entry.

E.51.3.11 static void BiometricEvaluation::IO::LogSheet::mergeLogSheets (vector< tr1::shared_ptr< LogSheet > > & logSheets) throw (Error::FileError, Error::StrategyError) [static]

Merge multiple LogSheets into a single [LogSheet](#).

LogSheets 2 - n will be appended to [LogSheet](#) 1.

Parameters

logSheets	LogSheets to merge.
-----------	---------------------

Exceptions

Error::FileError	Error during log sequence.
Error::StrategyError	Error during log sequence.

E.51.3.12 LogSheet& BiometricEvaluation::IO::LogSheet::operator= (const LogSheet &) [protected]

Prevent copying of [LogSheet](#) objects

E.51.3.13 `void BiometricEvaluation::IO::LogSheet::updateCursor () throw (Error::FileError)`
`[protected]`

Update the cursor position of the sequence file.

Exceptions

<i>Error::FileError</i>	Error getting file position from sequence file.
---	---

E.51.4 Member Data Documentation

E.51.4.1 `const char BiometricEvaluation::IO::LogSheet::CommentDelimiter = '#'` `[static]`

Delimiter for a comment line in the log sheet.

E.51.4.2 `const char BiometricEvaluation::IO::LogSheet::EntryDelimiter = 'E'` `[static]`

Delimiter for an entry line in the log sheet.

E.51.4.3 `const string BiometricEvaluation::IO::LogSheet::DescriptionTag` `[static]`

The tag for the description string.

E.51.4.4 `const int32_t BiometricEvaluation::IO::LogSheet::BE_LOGSHEET_SEQ_START = 1`
`[static]`

Sequence from beginning

E.51.4.5 `const int32_t BiometricEvaluation::IO::LogSheet::BE_LOGSHEET_SEQ_NEXT = 2`
`[static]`

Sequence from current position

E.51.4.6 `uint32_t BiometricEvaluation::IO::LogSheet::_entryNumber` `[protected]`

Number of the current entry

E.51.4.7 `auto_ptr<std::fstream> BiometricEvaluation::IO::LogSheet::_theLogFile` `[protected]`

Stream used for writing the log file

E.51.4.8 `bool BiometricEvaluation::IO::LogSheet::_autoSync` `[protected]`

Whether or not to [sync\(\)](#) on [write\(\)](#)

E.51.4.9 `tr1::shared_ptr<std::fstream>` `BiometricEvaluation::IO::LogSheet::_sequenceFile` [protected]

Stream used for sequencing

E.51.4.10 `streamoff` `BiometricEvaluation::IO::LogSheet::_cursor` [protected]

Position of the sequencer, relative to SOF

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

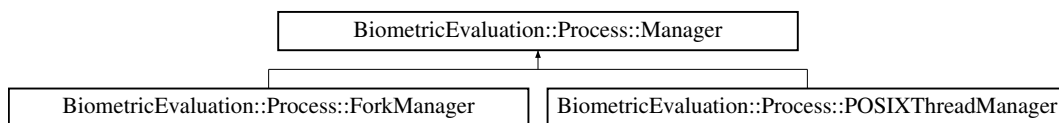
- `be_io_logsheet.h`

E.52 BiometricEvaluation::Process::Manager Class Reference

An interface for intranode process management classes.

```
#include <be_process_manager.h>
```

Inheritance diagram for `BiometricEvaluation::Process::Manager`:



Public Member Functions

- [Manager](#) ()
Manager constructor.
- virtual `tr1::shared_ptr < WorkerController > addWorker` (`tr1::shared_ptr< Worker > worker`)=0
Adds a Worker to be managed by this Manager.
- virtual `uint32_t getNumCompletedWorkers` () const =0 throw (`Error::StrategyError`)
Obtain the number of Workers that have exited.
- virtual `uint32_t getNumActiveWorkers` () const =0 throw (`Error::StrategyError`)
Obtain the number of Workers that are still working.
- virtual `uint32_t getTotalWorkers` () const =0
Obtain the number of Workers this class is handling.
- virtual void `startWorkers` (bool wait=true)=0 throw (`Error::ObjectExists`, `Error::StrategyError`)
Begin Worker's work.
- virtual void `startWorker` (`tr1::shared_ptr< WorkerController > worker`, bool wait=true)=0 throw (`Error::ObjectExists`, `Error::StrategyError`)
Start a Worker.
- virtual void `reset` ()=0 throw (`Error::ObjectExists`)
Reuse all Workers.
- virtual `int32_t stopWorker` (`tr1::shared_ptr< WorkerController > worker`)=0 throw (`Error::ObjectDoesNotExist`, `Error::StrategyError`)

Ask [Worker](#) to return as soon as possible.

- virtual [~Manager](#) ()
[Manager](#) destructor.

Protected Member Functions

- virtual void [_wait](#) ()=0
Do not return until all spawned processes exited.

E.52.1 Detailed Description

An interface for intranode process management classes.

E.52.2 Member Function Documentation

E.52.2.1 virtual `tr1::shared_ptr<WorkerController> BiometricEvaluation::Process::Manager::addWorker (tr1::shared_ptr< Worker > worker)` [pure virtual]

Adds a [Worker](#) to be managed by this [Manager](#).

Parameters

<i>worker</i>	A Worker instance to run.
---------------	---

Returns

`shared_ptr` to worker.

Implemented in [BiometricEvaluation::Process::POSIXThreadManager](#), and [BiometricEvaluation::Process::ForkManager](#).

E.52.2.2 virtual `uint32_t BiometricEvaluation::Process::Manager::getNumCompletedWorkers ()` const throw (`Error::StrategyError`) [pure virtual]

Obtain the number of Workers that have exited.

Returns

The number of Workers that have exited.

Exceptions

Error::StrategyError	No Workers have started working yet.
--------------------------------------	--------------------------------------

Implemented in [BiometricEvaluation::Process::POSIXThreadManager](#), and [BiometricEvaluation::Process::ForkManager](#).

E.52.2.3 `virtual uint32_t BiometricEvaluation::Process::Manager::getNumActiveWorkers () const throw (Error::StrategyError) [pure virtual]`

Obtain the number of Workers that are still working.

Returns

The number of Workers that are still working.

Exceptions

<i>Error::StrategyError</i>	No Workers have started working yet.
---	--------------------------------------

Implemented in [BiometricEvaluation::Process::POSIXThreadManager](#), and [BiometricEvaluation::Process::ForkManager](#).

E.52.2.4 `virtual uint32_t BiometricEvaluation::Process::Manager::getTotalWorkers () const [pure virtual]`

Obtain the number of Workers this class is handling.

Returns

Number of Workers.

Implemented in [BiometricEvaluation::Process::POSIXThreadManager](#), and [BiometricEvaluation::Process::ForkManager](#).

E.52.2.5 `virtual void BiometricEvaluation::Process::Manager::startWorkers (bool wait = true) throw (Error::ObjectExists, Error::StrategyError) [pure virtual]`

Begin Worker's work.

Parameters

<code>in</code>	<code>wait</code>	Whether or not to wait for all Workers to return before returning.
-----------------	-------------------	--

Exceptions

<i>Error::ObjectExists</i>	At least one Worker is already working.
<i>Error::StrategyError</i>	Problem starting Workers.

Implemented in [BiometricEvaluation::Process::POSIXThreadManager](#), and [BiometricEvaluation::Process::ForkManager](#).

E.52.2.6 `virtual void BiometricEvaluation::Process::Manager::startWorker (tr1::shared_ptr< WorkerController > worker, bool wait = true) throw (Error::ObjectExists, Error::StrategyError) [pure virtual]`

Start a [Worker](#).

Parameters

<i>worker</i>	Pointer to a WorkerController that is being managed by this Manager instance.
<i>wait</i>	Whether or not to wait for this Worker to exit before returning control to the caller.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

Note

Some implementations of this interface may call the system exit function from this routine. Therefore, the application's implementation of `workerMain()` should release all resources before returning.

Implemented in [BiometricEvaluation::Process::POSIXThreadManager](#), and [BiometricEvaluation::Process::ForkManager](#).

E.52.2.7 `virtual void BiometricEvaluation::Process::Manager::reset () throw (Error::ObjectExists) [pure virtual]`

Reuse all Workers.

Exceptions

Error::ObjectExists	At least one Worker is still working.
-------------------------------------	---

Implemented in [BiometricEvaluation::Process::ForkManager](#), and [BiometricEvaluation::Process::POSIXThreadManager](#).

E.52.2.8 `virtual int32_t BiometricEvaluation::Process::Manager::stopWorker (tr1::shared_ptr< WorkerController > worker) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]`

Ask [Worker](#) to return as soon as possible.

Parameters

<i>worker</i>	Pointer to the WorkerController that should be stopped.
---------------	---

Returns

Return code of worker.

Exceptions

<i>Error::ObjectDoesNotExist</i>	worker is not working.
<i>Error::StrategyError</i>	Problem asking worker to stop.

Implemented in [BiometricEvaluation::Process::ForkManager](#), and [BiometricEvaluation::Process::POSIXThreadManager](#).

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

- `be_process_manager.h`

E.53 BiometricEvaluation::IO::ManifestEntry Struct Reference

```
#include <be_io_archiverecstore.h>
```

Public Attributes

- long [offset](#)
- uint64_t [size](#)

E.53.1 Detailed Description

Info about a single archive element

E.53.2 Member Data Documentation

E.53.2.1 long BiometricEvaluation::IO::ManifestEntry::offset

The offset from the beginning of the file/memory

E.53.2.2 uint64_t BiometricEvaluation::IO::ManifestEntry::size

The length from offset this element spans

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

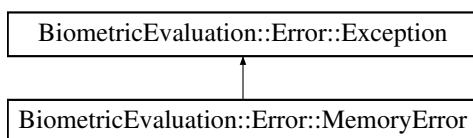
- `be_io_archiverecstore.h`

E.54 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.54.1 Detailed Description

An error occurred when allocating an object.

E.54.2 Constructor & Destructor Documentation

E.54.2.1 [BiometricEvaluation::Error::MemoryError::MemoryError](#) ()

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

Returns

The [MemoryError](#) object.

E.54.2.2 [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:

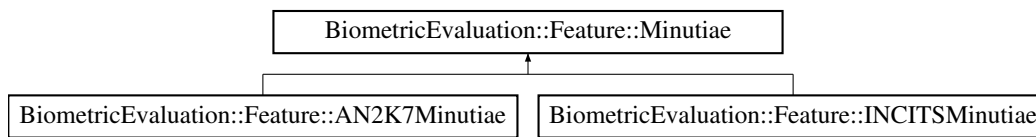
- `be_error_exception.h`

E.55 [BiometricEvaluation::Feature::Minutiae](#) Class Reference

A class to represent a set of minutiae data points.

```
#include <be_feature_minutiae.h>
```

Inheritance diagram for [BiometricEvaluation::Feature::Minutiae](#):



Public Member Functions

- virtual MinutiaeFormat::Kind [getFormat](#) () const =0
Obtain the minutiae format kind.
- virtual MinutiaPointSet [getMinutiaPoints](#) () const =0
Obtain the set of finger minutiae data points. The set may be empty.
- virtual RidgeCountItemSet [getRidgeCountItems](#) () const =0
Obtain the set of ridge count data items. The set may be empty.
- virtual CorePointSet [getCores](#) () const =0
Obtains the set of core positions. The set may be empty.
- virtual DeltaPointSet [getDeltas](#) () const =0
Obtains the set of delta positions. The set may be empty.

E.55.1 Detailed Description

A class to represent a set of minutiae data points.

Each set includes the core and delta data points, if they are included in the source record. This class represents an interface that subclasses of this class will implement, providing more information on the minutiae that is specific to the record format represented by that class.

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

- `be_feature_minutiae.h`

E.56 BiometricEvaluation::Feature::MinutiaeFormat Class Reference

Enumerate the minutiae format standards.

```
#include <be_feature_minutiae.h>
```

Public Types

- enum **Kind** { AN2K7 = 0, IAFIS, Cogent, Motorola, Sagem, NEC, Identix, M1 }

E.56.1 Detailed Description

Enumerate the minutiae format standards.

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

- `be_feature_minutiae.h`

E.57 BiometricEvaluation::Feature::MinutiaeType Class Reference

Enumerate the types of minutiae: Ridge Ending, Bifurcation, Compound, or other.

```
#include <be_feature_minutiae.h>
```

Public Types

- enum **Kind** { **RidgeEnding** = 0, **Bifurcation**, **Compound**, **Other** }

E.57.1 Detailed Description

Enumerate the types of minutiae: Ridge Ending, Bifurcation, Compound, or other.

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

- be_feature_minutiae.h

E.58 BiometricEvaluation::Feature::MinutiaPoint Struct Reference

Representation of a finger minutiae data point.

```
#include <be_feature_minutiae.h>
```

Public Attributes

- unsigned int **index**
- bool **has_type**
- MinutiaeType::Kind **type**
- [Image::Coordinate](#) **coordinate**
- unsigned int **theta**
- bool **has_quality**
- unsigned int **quality**

E.58.1 Detailed Description

Representation of a finger minutiae data point.

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

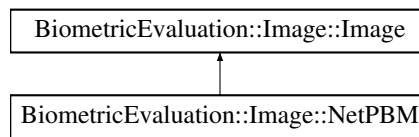
- be_feature_minutiae.h

E.59 BiometricEvaluation::Image::NetPBM Class Reference

A NetPBM-encoded image.

```
#include <be_image_netpbm.h>
```

Inheritance diagram for BiometricEvaluation::Image::NetPBM:



Public Types

- enum **Kind** { **ASCIIPortableBitmap** = 1, **ASCIIPortableGraymap** = 2, **ASCIIPortablePixmap** = 3, **BinaryPortableBitmap** = 4, **BinaryPortableGraymap** = 5, **BinaryPortablePixmap** = 6 }

Public Member Functions

- **NetPBM** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- **Memory::AutoArray**< uint8_t > **getRawData** () const throw (Error::DataError)
Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::AutoArray**< uint8_t > **getRawGrayscaleData** (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)
Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool **isNetPBM** (const uint8_t *data, const size_t size)
- static void **skipLine** (**Memory::uint8Array** &data, size_t &offset) throw (out_of_range)
Skip an entire line of input, placing offset at the first character after the newline.
- static void **skipComment** (**Memory::uint8Array** &data, size_t &offset) throw (out_of_range)
Skip a block of comments in input.
- static string **getNextValue** (**Memory::uint8Array** &data, size_t &offset, size_t sizeofValue=0)
Obtain the next space-separated value from data, beginning at offset.
- static **Memory::uint8Array** **ASCIIBitmapTo8Bit** (**Memory::uint8Array** &bitmap, uint32_t width, uint32_t height) throw (out_of_range)
Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.
- static **Memory::uint8Array** **ASCIIPixmapToBinaryPixmap** (**Memory::uint8Array** &ASCIIBuf, uint32_t width, uint32_t height, uint8_t depth, uint32_t maxColor) throw (out_of_range, Error::ParameterError)
Convert an ASCII pixel map buffer into a binary pixel map buffer.
- static **Memory::uint8Array** **BinaryBitmapTo8Bit** (**Memory::uint8Array** &bitmap, uint32_t width, uint32_t height) throw (out_of_range)
Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

E.59.1 Detailed Description

A NetPBM-encoded image.

Note

While a [NetPBM](#) file can contain more than one image, this class will only support the first image found in any file, also known as the "plain" [NetPBM](#) format.

E.59.2 Member Function Documentation**E.59.2.1** `Memory::AutoArray<uint8_t> BiometricEvaluation::Image::NetPBM::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

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

Note

The raw data returned from this method is encoded at the same bit depth as the compressed data, except in the case of 1-bit (bitmap) images, which are expanded to 8-bit.

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

E.59.2.2 `Memory::AutoArray<uint8_t> BiometricEvaluation::Image::NetPBM::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

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

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. *depth* adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

E.59.2.3 `static bool BiometricEvaluation::Image::NetPBM::isNetPBM (const uint8_t * data, const size_t size)`
[static]

Whether or not *data* is a netpbm image.

Parameters

<i>in</i>	<i>data</i>	The buffer to check.
<i>in</i>	<i>size</i>	The size of <i>data</i> .

Returns

true if *data* appears to be a netpbm image, false otherwise.

E.59.2.4 `static void BiometricEvaluation::Image::NetPBM::skipLine (Memory::uint8Array & data, size_t & offset) throw (out_of_range)` [static]

Skip an entire line of input, placing *offset* at the first character after the newline.

Parameters

<i>data</i>	Buffer with line to be skipped.
<i>offset</i>	Position within <i>data</i> from which the rest of the line should be read.

Exceptions

<i>out_of_range</i>	End of line not encountered before end of <i>data</i> or on last line of <i>data</i> .
---------------------	--

E.59.2.5 `static void BiometricEvaluation::Image::NetPBM::skipComment (Memory::uint8Array & data, size_t & offset) throw (out_of_range)` [static]

Skip a block of comments in input.

Parameters

<i>data</i>	Buffer with comment to be skipped.
<i>offset</i>	Position within <i>data</i> from which the rest of the line should be read.

Exceptions

<i>out_of_range</i>	End of line not encountered before end of <i>data</i> or on last line of <i>data</i> .
---------------------	--

E.59.2.6 `static string BiometricEvaluation::Image::NetPBM::getNextValue (Memory::uint8Array & data, size_t & offset, size_t sizeOfValue = 0) [static]`

Obtain the next space-separated value from data, beginning at offset.

Parameters

<i>data</i>	Buffer where next value will be obtained.
<i>offset</i>	Current starting position within data.
<i>sizeOfValue</i>	In the event that the values in data are not space-separated, return a value when it reaches sizeOfValue length. 0 assumes space-separated.

Returns

Next value from data.

E.59.2.7 `static Memory::uint8Array BiometricEvaluation::Image::NetPBM::ASCIIBitmapTo8Bit (Memory::uint8Array & bitmap, uint32_t width, uint32_t height) throw (out_of_range) [static]`

Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Parameters

<i>bitmap</i>	Bitmap data buffer.
<i>width</i>	Width of image in bitmap.
<i>height</i>	Height of image in bitmap.

Returns

8-bit depth representation of bitmap

Exceptions

<i>out_of_range</i>	Error extracting a value from the bitmap.
---------------------	---

E.59.2.8 `static Memory::uint8Array BiometricEvaluation::Image::NetPBM::ASCIIPixmapToBinaryPixmap (Memory::uint8Array & ASCIIBuf, uint32_t width, uint32_t height, uint8_t depth, uint32_t maxColor) throw (out_of_range, Error::ParameterError) [static]`

Convert an ASCII pixel map buffer into a binary pixel map buffer.

Parameters

<i>ASCIIBuf</i>	ASCII pixel map data buffer.
<i>width</i>	Width of image in pixel map.
<i>height</i>	Height of image in pixel map.
<i>depth</i>	Depth of image in pixel map.
<i>maxColor</i>	Maximum color value per pixel. Intensities will be scaled based on this value.

Returns

Binary pixel map representation of the ASCII pixel map in the same depth as the original.

Exceptions

<i>out_of_range</i>	Error extracting a value from the pixel map.
<i>Error::ParameterError</i>	Invalid value for depth, must be a multiple of Image::bitsPerComponent .

E.59.2.9 static `Memory::uint8Array BiometricEvaluation::Image::NetPBM::BinaryBitmapTo8Bit (Memory::uint8Array & bitmap, uint32_t width, uint32_t height) throw (out_of_range) [static]`

Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Parameters

<i>bitmap</i>	Bitmap data buffer.
<i>width</i>	Width of image in bitmap.
<i>height</i>	Height of image in bitmap.

Returns

8-bit depth representation of bitmap

Exceptions

<i>out_of_range</i>	Error extracting a value from the bitmap.
---------------------	---

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

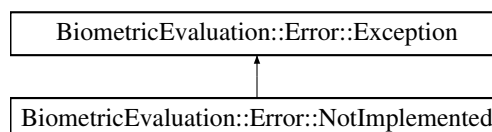
- `be_image_netpbm.h`

E.60 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.60.1 Detailed Description

A [NotImplemented](#) object is thrown when the underlying implementation of this interface has not or could not be created.

E.60.2 Constructor & Destructor Documentation

E.60.2.1 BiometricEvaluation::Error::NotImplemented::NotImplemented ()

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

Returns

The [NotImplemented](#) object.

E.60.2.2 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:

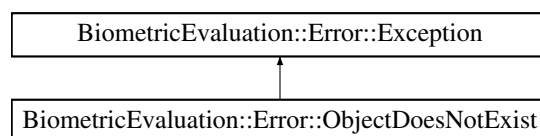
- be_error_exception.h

E.61 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.61.1 Detailed Description

The named object does not exist.

E.61.2 Constructor & Destructor Documentation

E.61.2.1 BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ()

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

Returns

The [ObjectDoesNotExist](#) object.

E.61.2.2 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:

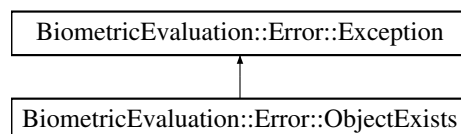
- be_error_exception.h

E.62 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.62.1 Detailed Description

The named object exists and will not be replaced.

E.62.2 Constructor & Destructor Documentation

E.62.2.1 BiometricEvaluation::Error::ObjectExists::ObjectExists ()

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

Returns

The [ObjectExists](#) object.

E.62.2.2 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:

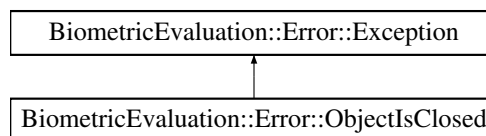
- be_error_exception.h

E.63 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.63.1 Detailed Description

The object is closed.

E.63.2 Constructor & Destructor Documentation

E.63.2.1 BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ()

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

Returns

The [ObjectIsClosed](#) object.

E.63.2.2 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:

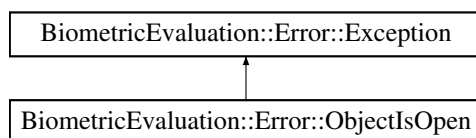
- `be_error_exception.h`

E.64 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.64.1 Detailed Description

The object is already opened.

E.64.2 Constructor & Destructor Documentation

E.64.2.1 BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ()

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

Returns

The [ObjectIsOpen](#) object.

E.64.2.2 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:

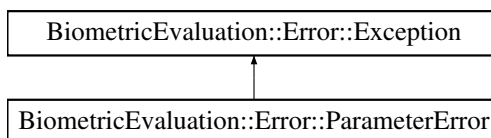
- `be_error_exception.h`

E.65 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.65.1 Detailed Description

An invalid parameter was passed to a constructor or method.

E.65.2 Constructor & Destructor Documentation

E.65.2.1 BiometricEvaluation::Error::ParameterError::ParameterError ()

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

Returns

The [ParameterError](#) object.

E.65.2.2 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.66 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification Class Reference

Pattern classification codes.

```
#include <be_feature_an2k7minutiae.h>
```

Classes

- struct [Entry](#)

Public Types

- typedef struct [Entry](#) **Entry**

E.66.1 Detailed Description

Pattern classification codes.

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

- [be_feature_an2k7minutiae.h](#)

E.67 BiometricEvaluation::Finger::PatternClassification Class Reference

Pattern classification codes.

```
#include <be_finger.h>
```

Public Types

- enum **Kind** { **PlainArch** = 0, **TentedArch**, **RadialLoop**, **UlnarLoop**, **PlainWhorl**, **CentralPocket-Loop**, **DoubleLoop**, **AccidentalWhorl**, **Whorl**, **RightSlantLoop**, **LeftSlantLoop**, **Scar**, **Amputation**, **Unknown** }

E.67.1 Detailed Description

Pattern classification codes.

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

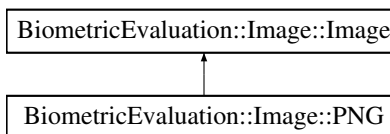
- `be_finger.h`

E.68 BiometricEvaluation::Image::PNG Class Reference

A PNG-encoded image.

```
#include <be_image_png.h>
```

Inheritance diagram for BiometricEvaluation::Image::PNG:



Classes

- struct **png_buffer**
Wrapper for reading PNG-encoded data from an AutoArray with libpng.

Public Member Functions

- **PNG** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- **Memory::AutoArray**< uint8_t > **getRawData** () const throw (Error::DataError)
Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::AutoArray**< uint8_t > **getRawGrayscaleData** (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)
Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool [isPNG](#) (const uint8_t *data)

E.68.1 Detailed Description

A PNG-encoded image.

E.68.2 Member Function Documentation

E.68.2.1 `Memory::AutoArray<uint8_t> BiometricEvaluation::Image::PNG::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

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

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

E.68.2.2 `Memory::AutoArray<uint8_t> BiometricEvaluation::Image::PNG::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

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

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

E.68.2.3 `static bool BiometricEvaluation::Image::PNG::isPNG (const uint8_t* data)` `[static]`

Whether or not data is a [PNG](#) image.

Parameters

<code>in</code>	<code><i>data</i></code>	The buffer to check.
-----------------	--------------------------	----------------------

Returns

true if data appears to be a [PNG](#) image, false otherwise

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

- `be_image_png.h`

E.69 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.69.1 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:

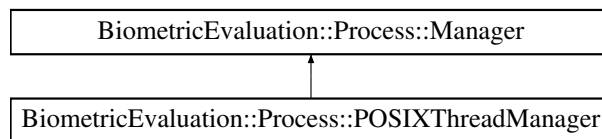
- `be_finger.h`

E.70 BiometricEvaluation::Process::POSIXThreadManager Class Reference

[Manager](#) implementation that starts Workers in POSIX threads.

```
#include <be_process_posixthreadmanager.h>
```

Inheritance diagram for BiometricEvaluation::Process::POSIXThreadManager:



Public Member Functions

- [POSIXThreadManager](#) ()
- `uint32_t` [getNumCompletedWorkers](#) () const throw (Error::StrategyError)
Obtain the number of Workers that have exited.
- `uint32_t` [getNumActiveWorkers](#) () const throw (Error::StrategyError)
Obtain the number of Workers that are still running.
- `uint32_t` [getTotalWorkers](#) () const
Obtain the number of Workers this class is handling.
- `tr1::shared_ptr< WorkerController >` [addWorker](#) (tr1::shared_ptr< [Worker](#) > worker)
Adds a [Worker](#) to be managed by this [Manager](#).
- void [startWorkers](#) (bool wait=true) throw (Error::ObjectExists, Error::StrategyError)
Begin Worker's work.
- void [startWorker](#) (tr1::shared_ptr< [WorkerController](#) > worker, bool wait=true) throw (Error::ObjectExists, Error::StrategyError)
Start a [Worker](#).
- `int32_t` [stopWorker](#) (tr1::shared_ptr< [WorkerController](#) > workerController) throw (Error::ObjectDoesNotExist, Error::StrategyError)
Ask [Worker](#) to exit.
- void [reset](#) () throw (Error::ObjectExists)
Reuse all Workers.
- [~POSIXThreadManager](#) ()
~POSIXThreadManager destructor.

Protected Attributes

- `vector< tr1::shared_ptr < POSIXThreadWorkerController > >` [_workers](#)

E.70.1 Detailed Description

[Manager](#) implementation that starts Workers in POSIX threads.

E.70.2 Constructor & Destructor Documentation

E.70.2.1 BiometricEvaluation::Process::POSIXThreadManager::POSIXThreadManager ()

[POSIXThreadManager](#) constructor.

E.70.3 Member Function Documentation

E.70.3.1 uint32_t BiometricEvaluation::Process::POSIXThreadManager::getNumCompletedWorkers () const throw (Error::StrategyError) [virtual]

Obtain the number of Workers that have exited.

Returns

The number of Workers that have exited.

Exceptions

Error::StrategyError	No processes have been spawned yet.
--------------------------------------	-------------------------------------

Implements [BiometricEvaluation::Process::Manager](#).

E.70.3.2 uint32_t BiometricEvaluation::Process::POSIXThreadManager::getNumActiveWorkers () const throw (Error::StrategyError) [virtual]

Obtain the number of Workers that are still running.

Returns

The number of Workers that are still running.

Exceptions

Error::StrategyError	No Workers have been spawned yet.
--------------------------------------	-----------------------------------

Implements [BiometricEvaluation::Process::Manager](#).

E.70.3.3 uint32_t BiometricEvaluation::Process::POSIXThreadManager::getTotalWorkers () const [virtual]

Obtain the number of Workers this class is handling.

Returns

Number of Workers.

Implements [BiometricEvaluation::Process::Manager](#).

E.70.3.4 `tr1::shared_ptr<WorkerController> BiometricEvaluation::Process::POSIXThreadManager::add-Worker (tr1::shared_ptr< Worker > worker) [virtual]`

Adds a [Worker](#) to be managed by this [Manager](#).

Parameters

<i>worker</i>	A Worker instance to run.
---------------	---

Returns

`shared_ptr` to `worker`.

Implements [BiometricEvaluation::Process::Manager](#).

E.70.3.5 `void BiometricEvaluation::Process::POSIXThreadManager::startWorkers (bool wait = true) throw (Error::ObjectExists, Error::StrategyError) [virtual]`

Begin Worker's work.

Parameters

<i>in</i>	<i>wait</i>	Whether or not to wait for all Workers to return before returning.
-----------	-------------	--

Exceptions

Error::ObjectExists	One or more of the Workers is already working.
Error::StrategyError	Problem starting the Worker .

Implements [BiometricEvaluation::Process::Manager](#).

E.70.3.6 `void BiometricEvaluation::Process::POSIXThreadManager::startWorker (tr1::shared_ptr< WorkerController > worker, bool wait = true) throw (Error::ObjectExists, Error::StrategyError) [virtual]`

Start a [Worker](#).

Parameters

<i>worker</i>	Pointer to a WorkerController that is being managed by this Manager instance.
<i>wait</i>	Whether or not to wait for this Worker to exit before returning control to the caller.

Exceptions

Error::ObjectExists	<code>worker</code> is already working.
Error::StrategyError	<code>worker</code> is not managed by this Manager instance.

Implements [BiometricEvaluation::Process::Manager](#).

E.70.3.7 `int32_t BiometricEvaluation::Process::POSIXThreadManager::stopWorker (tr1::shared_ptr< WorkerController > workerController) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Ask [Worker](#) to exit.

Parameters

<i>worker-Controller</i>	Pointer to the WorkerController that should be stopped.
--------------------------	---

Returns

Exit status of worker.

Exceptions

<i>Error::ObjectDoesNotExist</i>	worker is not working.
<i>Error::StrategyError</i>	Problem sending the signal.

Implements [BiometricEvaluation::Process::Manager](#).

E.70.3.8 `void BiometricEvaluation::Process::POSIXThreadManager::reset () throw (Error::ObjectExists) [virtual]`

Reuse all Workers.

Exceptions

<i>Error::ObjectExists</i>	At least one Worker is still working.
----------------------------	---

Implements [BiometricEvaluation::Process::Manager](#).

E.70.4 Member Data Documentation

E.70.4.1 `vector< tr1::shared_ptr<POSIXThreadWorkerController> > BiometricEvaluation::Process::-POSIXThreadManager::_workers [protected]`

Workers that have been added

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

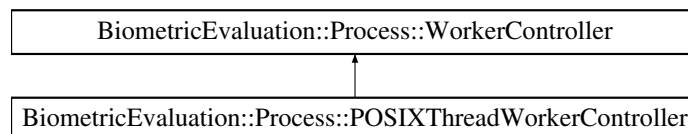
- `be_process_posixthreadmanager.h`

E.71 BiometricEvaluation::Process::POSIXThreadWorkerController Class Reference

Decorated [Worker](#) returned from a [Process::POSIXThreadManager](#).

```
#include <be_process_posixthreadmanager.h>
```

Inheritance diagram for BiometricEvaluation::Process::POSIXThreadWorkerController:



Public Member Functions

- void [reset](#) () throw (Error::ObjectExists)
Reuse the [Worker](#).
- bool [isWorking](#) () const
Obtain whether or not [Worker](#) is working.
- [~POSIXThreadWorkerController](#) ()
POSIXThreadWorkerController destructor.

Friends

- class [POSIXThreadManager](#)

E.71.1 Detailed Description

Decorated [Worker](#) returned from a [Process::POSIXThreadManager](#).

E.71.2 Member Function Documentation

E.71.2.1 void BiometricEvaluation::Process::POSIXThreadWorkerController::reset () throw (Error::ObjectExists) [virtual]

Reuse the [Worker](#).

Exceptions

Error::ObjectExists	The previously started Worker is still running.
-------------------------------------	---

Reimplemented from [BiometricEvaluation::Process::WorkerController](#).

E.71.2.2 bool BiometricEvaluation::Process::POSIXThreadWorkerController::isWorking () const [virtual]

Obtain whether or not [Worker](#) is working.

Returns

Whether or not the [Worker](#) is working.

Implements [BiometricEvaluation::Process::WorkerController](#).

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

- `be_process_posixthreadmanager.h`

E.72 BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPosition-Coordinate Struct Reference

Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

```
#include <be_finger_an2kview_varres.h>
```

Public Member Functions

- [PrintPositionCoordinate](#) (FingerImageCode::Kind &[fingerView](#), FingerImageCode::Kind &[segment](#), - Image::CoordinateSet &[coordinates](#))
Construct a [PrintPositionCoordinate](#).

Public Attributes

- FingerImageCode::Kind [fingerView](#)
- FingerImageCode::Kind [segment](#)
- Image::CoordinateSet [coordinates](#)

E.72.1 Detailed Description

Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

E.72.2 Constructor & Destructor Documentation

E.72.2.1 BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::PrintPositionCoordinate (FingerImageCode::Kind & *fingerView*, FingerImageCode::Kind & *segment*, Image::CoordinateSet & *coordinates*)

Construct a [PrintPositionCoordinate](#).

Parameters

<i>fingerView</i>	The full finger view being referred to.
<i>segment</i>	Location of a segment within <i>fingerView</i> . If segment is NA, the image referred to is the entire image or tip.
<i>coordinates</i>	Two coordinates creating a bounding rectangle (top left vertex, lower right vertex).

E.72.3 Member Data Documentation

E.72.3.1 FingerImageCode::Kind BiometricEvaluation::Finger::AN2KViewVariableResolution::Print-PositionCoordinate::fingerView

Full finger view being bounded

E.72.3.2 FingerImageCode::Kind BiometricEvaluation::Finger::AN2KViewVariableResolution::Print-PositionCoordinate::segment

Segment within full finger view bound

E.72.3.3 Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewVariableResolution::Print-PositionCoordinate::coordinates

Two coordinates forming bounding box

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

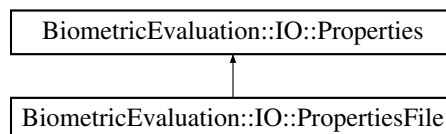
- be_finger_an2kview_varres.h

E.73 BiometricEvaluation::IO::Properties Class Reference

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

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

Inheritance diagram for BiometricEvaluation::IO::Properties:



Public Types

- typedef PropertiesMap::const_iterator [const_iterator](#)

Public Member Functions

- [Properties](#) (uint8_t mode=IO::READWRITE)
Construct a new [Properties](#) object.
- [Properties](#) (const uint8_t *buffer, const size_t size, uint8_t mode=IO::READWRITE) throw (Error::StrategyError)
Construct a new [Properties](#) object from the contents of a buffer.
- virtual void [setProperty](#) (const string &property, const string &value) throw (Error::StrategyError)

Set a property with a value.

- virtual void [setPropertyFromInteger](#) (const string &property, int64_t value) throw (Error::StrategyError)

Set a property with an integer value.

- virtual void [setPropertyFromDouble](#) (const string &property, double value) throw (Error::StrategyError)

Set a property with a double value.

- virtual void [removeProperty](#) (const string &property) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Remove a property.

- virtual string [getProperty](#) (const string &property) const throw (Error::ObjectDoesNotExist)

Retrieve a property value as a string object.

- virtual int64_t [getPropertyAsInteger](#) (const string &property) const throw (Error::ObjectDoesNotExist, Error::ConversionError)

Retrieve a property value as an integer value.

- virtual double [getPropertyAsDouble](#) (const string &property) const throw (Error::ObjectDoesNotExist)

Retrieve a property value as a double value.

- [const_iterator begin](#) () const

Obtain iterator to the first property.

- [const_iterator end](#) () const

Obtain iterator to one past the last property.

- virtual [~Properties](#) ()

Protected Member Functions

- uint8_t [getMode](#) () const

Obtain the mode of the [Properties](#) object.

- void [initWithBuffer](#) (const [Memory::uint8Array](#) &buffer) throw (Error::StrategyError)

Initialize the [PropertiesMap](#) with the contents of a properly formatted buffer.

- void [initWithBuffer](#) (const uint8_t *const buffer, size_t size) throw (Error::StrategyError)

Initialize the [PropertiesMap](#) with the contents of a properly formatted buffer.

E.73.1 Detailed Description

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

E.73.2 Member Typedef Documentation

E.73.2.1 `typedef PropertiesMap::const_iterator BiometricEvaluation::IO::Properties::const_iterator`

Convenience const iterator over a [Properties](#)

E.73.3 Constructor & Destructor Documentation

E.73.3.1 BiometricEvaluation::IO::Properties::Properties (uint8_t *mode* = IO::READWRITE)

Construct a new [Properties](#) object.

Parameters

<i>in</i>	<i>mode</i>	The read/write mode of the object.
-----------	-------------	------------------------------------

Returns

An object representing the properties set.

E.73.3.2 BiometricEvaluation::IO::Properties::Properties (const uint8_t * *buffer*, const size_t *size*, uint8_t *mode* = IO::READWRITE) throw (Error::StrategyError)

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

The format of the buffer can be seen in [PropertiesFile](#).

Parameters

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

Returns

An object representing the properties set.

Exceptions

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

E.73.3.3 virtual BiometricEvaluation::IO::Properties::~~Properties () [virtual]

Destructor

E.73.4 Member Function Documentation

E.73.4.1 virtual void BiometricEvaluation::IO::Properties::setProperty (const string & *property*, const string & *value*) throw (Error::StrategyError) [virtual]

Set a property with a value.

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

Parameters

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

Exceptions

<i>Error::StrategyError</i>	The <i>Properties</i> object is read-only.
---	--

E.73.4.2 `virtual void BiometricEvaluation::IO::Properties::setPropertyFromInteger (const string & property, int64_t value) throw (Error::StrategyError) [virtual]`

Set a property with an integer value.

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

Parameters

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

Exceptions

<i>Error::StrategyError</i>	The <i>Properties</i> object is read-only.
---	--

E.73.4.3 `virtual void BiometricEvaluation::IO::Properties::setPropertyFromDouble (const string & property, double value) throw (Error::StrategyError) [virtual]`

Set a property with a double value.

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

Parameters

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

Exceptions

<i>Error::StrategyError</i>	The <i>Properties</i> object is read-only.
---	--

E.73.4.4 `virtual void BiometricEvaluation::IO::Properties::removeProperty (const string & property) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Remove a property.

Parameters

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

Exceptions

<i>Error::ObjectDoesNotExist</i>	The named property does not exist.
<i>Error::StrategyError</i>	The Properties object is read-only.

E.73.4.5 virtual string BiometricEvaluation::IO::Properties::getProperty (const string & *property*) const throw (Error::ObjectDoesNotExist) [virtual]

Retrieve a property value as a string object.

Parameters

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

Exceptions

<i>Error::ObjectDoesNotExist</i>	The named property does not exist.
--	------------------------------------

E.73.4.6 virtual int64_t BiometricEvaluation::IO::Properties::getPropertyAsInteger (const string & *property*) const throw (Error::ObjectDoesNotExist, Error::ConversionError) [virtual]

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

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

E.73.4.7 virtual double BiometricEvaluation::IO::Properties::getPropertyAsDouble (const string & *property*) const throw (Error::ObjectDoesNotExist) [virtual]

Retrieve a property value as a double value.

Parameters

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

Exceptions

<i>Error::ObjectDoesNotExist</i>	The named property does not exist.
----------------------------------	------------------------------------

E.73.4.8 `const_iterator BiometricEvaluation::IO::Properties::begin () const`

Obtain iterator to the first property.

Returns

Iterator to first property.

E.73.4.9 `const_iterator BiometricEvaluation::IO::Properties::end () const`

Obtain iterator to one past the last property.

Returns

Iterator one past the last property.

E.73.4.10 `uint8_t BiometricEvaluation::IO::Properties::getMode () const` [protected]

Obtain the mode of the [Properties](#) object.

Returns

Mode (IO::READONLY or IO::READWRITE)

E.73.4.11 `void BiometricEvaluation::IO::Properties::initWithBuffer (const Memory::uint8Array & buffer) throw (Error::StrategyError)` [protected]

Initialize the PropertiesMap with the contents of a properly formatted buffer.

This method ensures that the PropertiesMap contains only the properties found within the buffer.

Parameters

<i>buffer</i>	Contents of a properties file.
---------------	--------------------------------

Exceptions

<i>Error::StrategyError</i>	A line of the buffer is malformed.
-----------------------------	------------------------------------

E.73.4.12 void BiometricEvaluation::IO::Properties::initWithBuffer (const uint8_t *const *buffer*, size_t *size*)
throw (Error::StrategyError) [protected]

Initialize the PropertiesMap with the contents of a properly formatted buffer.

This method ensures that the PropertiesMap contains only the properties found within the buffer.

Parameters

<i>buffer</i>	Contents of a properties file.
<i>size</i>	Size of the buffer.

Exceptions

<i>Error::StrategyError</i>	A line of the buffer is malformed.
---	------------------------------------

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

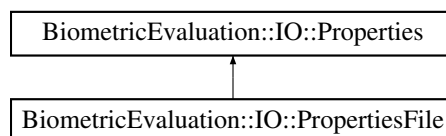
- be_io_properties.h

E.74 BiometricEvaluation::IO::PropertiesFile Class Reference

A [Properties](#) object persisted in an file on disk.

```
#include <be_io_propertiesfile.h>
```

Inheritance diagram for BiometricEvaluation::IO::PropertiesFile:



Public Member Functions

- [PropertiesFile](#) (const string &filename, uint8_t mode=IO::READWRITE) throw (Error::FileError, Error::StrategyError)

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

- void [sync](#) () throw (Error::FileError, Error::StrategyError)

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

- void [changeName](#) (const string &filename) throw (Error::StrategyError)

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

- [~PropertiesFile](#) ()

E.74.1 Detailed Description

A [Properties](#) object persisted in an file on disk.

An example file might look like this:

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

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

```
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.74.2 Constructor & Destructor Documentation

E.74.2.1 BiometricEvaluation::IO::PropertiesFile::PropertiesFile (const string & filename, uint8_t mode = IO::READWRITE) throw (Error::FileError, Error::StrategyError)

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

Parameters

in	<i>filename</i>	The name of the file to store the properties.
in	<i>mode</i>	The read/write mode of the object.

Returns

An object representing the properties set.

Exceptions

Error::StrategyError	A line in the properties file is malformed.
Error::FileError	An error occurred when using the underlying storage system.

E.74.2.2 BiometricEvaluation::IO::PropertiesFile::~~PropertiesFile ()

Destructor

E.74.3 Member Function Documentation

E.74.3.1 void BiometricEvaluation::IO::PropertiesFile::sync () throw (Error::FileError, Error::StrategyError)

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

Exceptions

Error::FileError	An error occurred when using the underlying storage system.
Error::StrategyError	The object was constructed with NULL as the file name, or is read-only.

E.74.3.2 void BiometricEvaluation::IO::PropertiesFile::changeName (const string & *filename*) throw (Error::StrategyError)

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

Note

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

Parameters

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

Exceptions

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

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

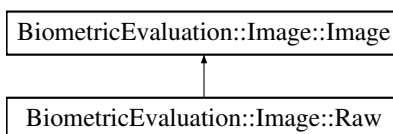
- be_io_propertiesfile.h

E.75 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::ParameterError)
Accessor for decompressed data in grayscale.

E.75.1 Detailed Description

An image with no encoding or compression.

E.75.2 Member Function Documentation

E.75.2.1 [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.75.2.2 [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

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

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

E.75.2.3 [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

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

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. *depth* adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

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

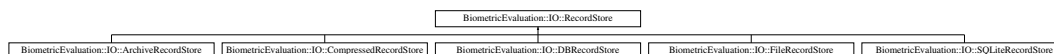
- `be_image_raw.h`

E.76 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 [insert](#) (const string &key, const [Memory::uint8Array](#) &data) throw (Error::ObjectExists, Error::StrategyError)
- virtual void [remove](#) (const string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t [read](#) (const string &key, void *const data) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t [read](#) (const string &key, [Memory::uint8Array](#) &data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
Read a complete record from a store.
- virtual void [replace](#) (const string &key, const void *const data, const uint64_t size)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void [replace](#) (const string &key, const [Memory::uint8Array](#) &data) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t [length](#) (const string &key) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void [flush](#) (const string &key) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t [sequence](#) (string &key, void *const data=NULL, int cursor=[BE_RECSTORE_SEQ_NEXT](#))=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
Sequence through a [RecordStore](#), returning the key/data pairs.
- virtual uint64_t [sequence](#) (string &key, [Memory::uint8Array](#) &data, int cursor=[BE_RECSTORE_SEQ_NEXT](#)) throw (Error::ObjectDoesNotExist, Error::StrategyError)
Sequence through a [RecordStore](#), returning the key/data pairs.
- virtual void [setCursorAtKey](#) (string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

Static Public Member Functions

- static tr1::shared_ptr < [RecordStore](#) > [openRecordStore](#) (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
Open an existing [RecordStore](#) and return a managed pointer to the the object representing that store.
- static tr1::shared_ptr < [RecordStore](#) > [createRecordStore](#) (const string &name, const string &description, const string &type, const string &destDir) throw (Error::ObjectExists, Error::StrategyError)
Create a new [RecordStore](#) and return a managed pointer to the the object representing that store.
- static void [removeRecordStore](#) (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- static void [mergeRecordStores](#) (const string &mergedName, const string &mergedDescription, const string &parentDir, const string &type, const vector< string > &path) throw (Error::ObjectExists, Error::StrategyError)
Create a new [RecordStore](#) that contains the contents of several other [RecordStores](#).

Static Public Attributes

- static const string [INVALIDKEYCHARS](#)
- static const char [KEY_SEGMENT_SEPARATOR](#) = '&'
- static const uint64_t [KEY_SEGMENT_START](#) = 1
- static const string [CONTROLFILENAME](#)
- static const string [NAMEPROPERTY](#)
- static const string [DESCRIPTIONPROPERTY](#)
- static const string [COUNTPROPERTY](#)
- static const string [TYPEPROPERTY](#)
- static const string [BERKELEYDBTYPE](#)
- static const string [ARCHIVETYPE](#)
- static const string [FILETYPE](#)
- static const string [SQLITETYPE](#)
- static const string [COMPRESSEDTYPE](#)
- static const string [DEFAULTTYPE](#)
- static const string [RSREADONLYERROR](#)
- 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
- void **setProperties** (const tr1::shared_ptr< [IO::Properties](#) > properties) throw (Error::StrategyError)
Replace existing [Properties](#) in [RecordStore](#) Control File.
- tr1::shared_ptr< [IO::Properties](#) > **getProperties** () const
Obtain a copy of the [Properties](#) object.

Static Protected Member Functions

- static string [genKeySegName](#) (const string &key, const uint64_t segnum)
Generate key segment names.

E.76.1 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.76.2 Constructor & Destructor Documentation

E.76.2.1 `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 RecordStore to be created.
in	<i>description</i>	The text used to describe the store.
in	<i>type</i>	The type of RecordStore .
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

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

E.76.2.2 `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 RecordStore has.

Returns

An object representing the existing store.

Exceptions

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

E.76.3 Member Function Documentation

E.76.3.1 string BiometricEvaluation::IO::RecordStore::getName () const

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

Returns

The RecordStore's name.

E.76.3.2 string BiometricEvaluation::IO::RecordStore::getDescription () const

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

Returns

The RecordStore's description.

E.76.3.3 unsigned int BiometricEvaluation::IO::RecordStore::getCount () const

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

Returns

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

E.76.3.4 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 RecordStore .
----	-------------	--

Exceptions

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

Reimplemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.5 virtual void BiometricEvaluation::IO::RecordStore::changeDescription (const string & *description*)
throw (Error::StrategyError) [virtual]

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

Parameters

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

Exceptions

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

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

E.76.3.6 virtual uint64_t BiometricEvaluation::IO::RecordStore::getSpaceUsed () const throw
(Error::StrategyError) [virtual]

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the [RecordStore](#).

Exceptions

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

Reimplemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.7 virtual void BiometricEvaluation::IO::RecordStore::sync () const throw (Error::StrategyError)
[virtual]

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.8 `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

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

Implemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.9 `virtual void BiometricEvaluation::IO::RecordStore::insert (const string & key, const Memory::uint8Array & data) throw (Error::ObjectExists, Error::StrategyError) [virtual]`

Insert a record into the store.

Parameters

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

Exceptions

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

E.76.3.10 `virtual void BiometricEvaluation::IO::RecordStore::remove (const string & key) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]`

Remove a record from the store.

Parameters

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

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::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.11 `virtual uint64_t BiometricEvaluation::IO::RecordStore::read (const string & key, void *const data) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]`

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

Parameters

in	<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

<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::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.12 `virtual uint64_t BiometricEvaluation::IO::RecordStore::read (const string & key, Memory::uint8Array & data) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

in	<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

<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.

E.76.3.13 `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::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.14 `virtual void BiometricEvaluation::IO::RecordStore::replace (const string & key, const Memory::uint8Array & data) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Replace a complete record in a [RecordStore](#).

Parameters

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

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.

E.76.3.15 `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::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.16 `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::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.17 `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 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

<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::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.18 `virtual uint64_t BiometricEvaluation::IO::RecordStore::sequence (string & key, Memory::uint8Array & data, int cursor = BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Sequence through a [RecordStore](#), returning the key/data pairs.

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

Parameters

out	<i>key</i>	The key of the currently sequenced record.
in	<i>data</i>	Pointer to where the data is to be written.
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

<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.

E.76.3.19 `virtual void BiometricEvaluation::IO::RecordStore::setCursorAtKey (string & key) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]`

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

Parameters

in	<i>key</i>	The key of the record which will be returned by the first subsequent call to sequence() .
----	------------	---

Exceptions

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

Implemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

E.76.3.20 `static tr1::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::openRecordStore (const string & name, const string & parentDir, uint8_t mode = READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]`

Open an existing [RecordStore](#) and return a managed pointer to the the object representing that store.

Applications can open existing record stores without the need to know what type of [RecordStore](#) it is.

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

Parameters

in	<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 RecordStore has.

Returns

An object representing the existing store.

Exceptions

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

E.76.3.21 `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

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

E.76.3.22 `static void BiometricEvaluation::IO::RecordStore::removeRecordStore (const string & name, const string & parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]`

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

Parameters

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

Exceptions

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

E.76.3.23 `static void BiometricEvaluation::IO::RecordStore::mergeRecordStores (const string & mergedName, const string & mergedDescription, const string & parentDir, const string & type, const vector< string > & path) throw (Error::ObjectExists, Error::StrategyError) [static]`

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

Parameters

in	<i>mergedName</i>	The name of the new RecordStore that will be created.
in	<i>mergedDescription</i>	The text used to describe the RecordStore .
in	<i>parentDir</i>	Where the new RecordStore should be rooted.

in	<i>type</i>	The type of RecordStore that mergedName should be.
in	<i>path</i>	Vector of string paths to RecordStores to open. These point to the RecordStores that will be merged.

Exceptions

Error::ObjectExists	A RecordStore with mergedNamed in parentDir already exists.
Error::StrategyError	An error occurred when using the underlying storage system.

E.76.3.24 static string BiometricEvaluation::IO::RecordStore::genKeySegName (const string & key, const uint64_t *segnum*) [static, protected]

Generate key segment names.

Parameters

<i>key</i>	Base key name.
<i>segnum</i>	Segment number for key (zero based).

Returns

Key segment name.

E.76.3.25 void BiometricEvaluation::IO::RecordStore::setProperties (const tr1::shared_ptr< IO::Properties > *properties*) throw (Error::StrategyError) [protected]

Replace existing [Properties](#) in [RecordStore](#) Control File.

Existing properties will be updated. [RecordStore](#) core properties will be ignored.

Parameters

in	<i>properties</i>	Shared pointer to Properties object.
----	-------------------	--

Exceptions

Error::StrategyError	RecordStore was opened READONLY.
--------------------------------------	--

E.76.3.26 tr1::shared_ptr<IO::Properties> BiometricEvaluation::IO::RecordStore::getProperties () const [protected]

Obtain a copy of the [Properties](#) object.

[RecordStore](#) core properties will be excluded.

Returns

Shared pointer to [Properties](#) object that may be modified.

E.76.4 Member Data Documentation

E.76.4.1 `const string BiometricEvaluation::IO::RecordStore::INVALIDKEYCHARS` [static]

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

E.76.4.2 `const char BiometricEvaluation::IO::RecordStore::KEY_SEGMENT_SEPARATOR = '&'`
[static]

Character used to separate key segments

E.76.4.3 `const uint64_t BiometricEvaluation::IO::RecordStore::KEY_SEGMENT_START = 1`
[static]

First segment number of a segmented record

E.76.4.4 `const string BiometricEvaluation::IO::RecordStore::CONTROLFILENAME` [static]

The name of the control file, a properties list

E.76.4.5 `const string BiometricEvaluation::IO::RecordStore::NAMEPROPERTY` [static]

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

E.76.4.6 `const string BiometricEvaluation::IO::RecordStore::DESCRIPTIONPROPERTY`
[static]

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

E.76.4.7 `const string BiometricEvaluation::IO::RecordStore::COUNTPROPERTY` [static]

Property key for the number of store items

E.76.4.8 `const string BiometricEvaluation::IO::RecordStore::TYPEPROPERTY` [static]

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

E.76.4.9 `const string BiometricEvaluation::IO::RecordStore::BERKELEYDBTYPE` [static]

[DBRecordStore](#) type

E.76.4.10 `const string BiometricEvaluation::IO::RecordStore::ARCHIVETYPE` `[static]`

[ArchiveRecordStore](#) type

E.76.4.11 `const string BiometricEvaluation::IO::RecordStore::FILETYPE` `[static]`

[FileRecordStore](#) type

E.76.4.12 `const string BiometricEvaluation::IO::RecordStore::SQLITETYPE` `[static]`

[SQLiteRecordStore](#) type

E.76.4.13 `const string BiometricEvaluation::IO::RecordStore::COMPRESSEDTYPE` `[static]`

[CompressedRecordStore](#) type

E.76.4.14 `const string BiometricEvaluation::IO::RecordStore::DEFAULTTYPE` `[static]`

Default [RecordStore](#)

E.76.4.15 `const string BiometricEvaluation::IO::RecordStore::RSREADONLYERROR` `[static]`

Message for READONLY [RecordStore](#) modification

E.76.4.16 `const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_START = 1`
`[static]`

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

E.76.4.17 `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.77 BiometricEvaluation::View::AN2KView::RecordType Class Reference

The type of AN2K record.

```
#include <be_view_an2kview.h>
```


Public Types

- enum **Kind** { **Type_1** = 1, **Type_2** = 2, **Type_3** = 3, **Type_4** = 4, **Type_5** = 5, **Type_6** = 6, **Type_7** = 7, **Type_8** = 8, **Type_9** = 9, **Type_10** = 10, **Type_11** = 11, **Type_12** = 12, **Type_13** = 13, **Type_14** = 14, **Type_15** = 15, **Type_16** = 16, **Type_17** = 17, **Type_99** = 99 }

E.77.1 Detailed Description

The type of AN2K record.

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

- `be_view_an2kview.h`

E.78 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.78.1 Detailed Description

A structure to represent the resolution of an image.

E.78.2 Member Enumeration Documentation

E.78.2.1 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.78.3 Constructor & Destructor Documentation**

E.78.3.1 `BiometricEvaluation::Image::Resolution::Resolution (const double xRes = 0.0, const double yRes = 0.0, const Kind units = PPI)`

Create a [Resolution](#) struct.

Parameters

<code>in</code>	<code><i>xRes</i></code>	Resolution along the X-axis
<code>in</code>	<code><i>yRes</i></code>	Resolution along the Y-axis
<code>in</code>	<code><i>units</i></code>	Units in which <code>xRes</code> and <code>yRes</code> are represented

E.78.4 Member Data Documentation

E.78.4.1 `double BiometricEvaluation::Image::Resolution::xRes`

[Resolution](#) along the X-axis

E.78.4.2 `double BiometricEvaluation::Image::Resolution::yRes`

[Resolution](#) along the Y-axis

E.78.4.3 `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.79 BiometricEvaluation::Feature::RidgeCountExtractionMethod Class Reference

Enumerate the types of extraction methods for ridge counts.

```
#include <be_feature_minutiae.h>
```

Public Types

- `enum Kind { NonSpecific = 0, FourNeighbor = 1, EightNeighbor = 2, Other = 3 }`

E.79.1 Detailed Description

Enumerate the types of extraction methods for ridge counts.

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

- `be_feature_minutiae.h`

E.80 BiometricEvaluation::Feature::RidgeCountItem Struct Reference

Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number.

```
#include <be_feature_minutiae.h>
```

Public Member Functions

- [RidgeCountItem](#) (RidgeCountExtractionMethod::Kind extraction_method, int index_one, int index_two, int count=0)
Create a [RidgeCountItem](#) struct.

Public Attributes

- RidgeCountExtractionMethod::Kind **extraction_method**
- int **index_one**
- int **index_two**
- int **count**

E.80.1 Detailed Description

Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number.

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

- `be_feature_minutiae.h`

E.81 BiometricEvaluation::Error::SignalManager Class Reference

A [SignalManager](#) object is used to handle signals that come from the operating system.

```
#include <be_error_signal_manager.h>
```

Public Member Functions

- [SignalManager](#) () throw (Error::StrategyError)
- [SignalManager](#) (const sigset_t signalSet) throw (Error::ParameterError)

- void [setSignalSet](#) (const sigset_t signalSet) throw (Error::ParameterError)
- void [clearSignalSet](#) ()
- void [setDefaultSignalSet](#) ()
- bool [sigHandled](#) ()
- void [start](#) () throw (Error::StrategyError)
- void [stop](#) () throw (Error::StrategyError)
- void [setSigHandled](#) ()
- void [clearSigHandled](#) ()

Static Public Attributes

- static bool [_canSigJump](#)
- static sigjmp_buf [_sigJumpBuf](#)

E.81.1 Detailed Description

A [SignalManager](#) object is used to handle signals that come from the operating system.

Applications typically do not invoke most methods of a [SignalManager](#), except the [setSignalSet\(\)](#), [setDefaultSignalSet\(\)](#), and [sigHandled\(\)](#). An application wishing to just catch memory errors can simply construct a [SignalManager](#) object, and invoke [sigHandled\(\)](#) at the end of the signal block to detect whether a signal was handled.

The BEGIN_SIGNAL_BLOCK macro sets up the jump block and tells the [SignalManager](#) object to start handling signals. Applications can call either [setSignalSet\(\)](#) or [setDefaultSignalSet\(\)](#) before invoking these macros to indicate which signals are to be handled.

The END_SIGNAL_BLOCK() macro clears the signal set, so from that point forward application code signals will be handled in the system's default manner until another signal block is created.

A [SignalManager](#) is passive (i.e. no signal handlers are installed) until that [start\(\)](#) method is called, and becomes passive when [stop\(\)](#) is invoked. The signals that are to be handled by the object are maintained as state, and the set of signals can be changed at any time, but are not in effect until [start\(\)](#) is called.

Attention

The [start\(\)](#), [stop\(\)](#), [setSigHandled\(\)](#) and [clearSigHandled\(\)](#) methods are not meant to be used directly by applications, which should use the BEGIN_SIGNAL_BLOCK()/END_SIGNAL_BLOCK() macro pair.

E.81.2 Constructor & Destructor Documentation

E.81.2.1 BiometricEvaluation::Error::SignalManager::SignalManager () throw (Error::StrategyError)

Construct a new [SignalManager](#) object with the default signal handling: SIGSEGV and SIGBUS.

Returns

The [SignalManager](#).

Exceptions

<i>Error::StrategyError</i>	Could not register the signal handler.
-----------------------------	--

E.81.2.2 BiometricEvaluation::Error::SignalManager::SignalManager (const sigset_t *signalSet*) throw (Error::ParameterError)

Construct a new [SignalManager](#) object with the specified signal handling, no defaults.

Parameters

<i>signalSet</i>	(in) The signal set; see sigaction(2), sigemptyset(3) and sigaddset(3).
------------------	---

Returns

The [SignalManager](#).

Exceptions

<i>Error::ParameterError</i>	One of the signals in <i>signalSet</i> cannot be handled (SIGKILL, SIGSTOP).
------------------------------	--

E.81.3 Member Function Documentation

E.81.3.1 void BiometricEvaluation::Error::SignalManager::setSignalSet (const sigset_t *signalSet*) throw (Error::ParameterError)

Set the signals this object will manage.

Parameters

<i>signalSet</i>	(in) The signal set; see sigaction(2), sigemptyset(3) and sigaddset(3).
------------------	---

Exceptions

<i>Error::ParameterError</i>	One of the signals in <i>signalSet</i> cannot be handled (SIGKILL, SIGSTOP).
------------------------------	--

E.81.3.2 void BiometricEvaluation::Error::SignalManager::clearSignalSet ()

Clear all signal handling.

E.81.3.3 void BiometricEvaluation::Error::SignalManager::setDefaultSignalSet ()

Set the default signals this object will manage: SIGSEGV and SIGBUS.

E.81.3.4 bool BiometricEvaluation::Error::SignalManager::sigHandled ()

Indicate whether a signal was handled.

Returns

true if a signal was handled, false otherwise.

E.81.3.5 void BiometricEvaluation::Error::SignalManager::start () throw (Error::StrategyError)

Start handling signals of the current signal set.

Exceptions

<i>Error::StrategyError</i>	Could not register the signal handler.
---	--

Note

If an application invokes [*start\(\)*](#) without setting up a signal jump block, behavior is undefined, and can result in an infinite loop if further processing causes a signal to be raised.

E.81.3.6 void BiometricEvaluation::Error::SignalManager::stop () throw (Error::StrategyError)

Stop handling signals of the current signal set.

Exceptions

<i>Error::StrategyError</i>	Could not register the signal handler.
---	--

E.81.3.7 void BiometricEvaluation::Error::SignalManager::setSigHandled ()

Set a flag to indicate a signal was handled.

E.81.3.8 void BiometricEvaluation::Error::SignalManager::clearSigHandled ()

Clear the indication that a signal was handled.

E.81.4 Member Data Documentation**E.81.4.1 bool BiometricEvaluation::Error::SignalManager::_canSigJump [static]**

Flag indicating can jump after handling a signal.

Note

Should not be directly used by applications.

E.81.4.2 sigjmp_buf BiometricEvaluation::Error::SignalManager::_sigJumpBuf [static]

The jump buffer used by the signal handler.

Note

Should not be directly used by applications.

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

- be_error_signal_manager.h

E.82 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.82.1 Detailed Description

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

E.82.2 Constructor & Destructor Documentation

E.82.2.1 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.82.3 Member Data Documentation

E.82.3.1 uint32_t BiometricEvaluation::Image::Size::xSize

Number of pixels on the X-axis

E.82.3.2 `uint32_t BiometricEvaluation::Image::Size::ySize`

Number of pixels on the Y-axis

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

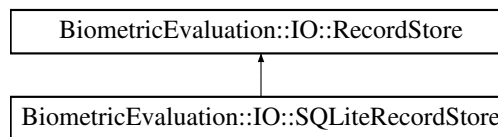
- `be_image.h`

E.83 `BiometricEvaluation::IO::SQLiteRecordStore` Class Reference

A [RecordStore](#) implementation using a SQLite database as the underlying record storage system.

```
#include <be_io_sqliterecstore.h>
```

Inheritance diagram for `BiometricEvaluation::IO::SQLiteRecordStore`:



Public Member Functions

- **SQLiteRecordStore** (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- **SQLiteRecordStore** (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void [changeName](#) (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- void [changeDescription](#) (const string &description) throw (Error::StrategyError)
- uint64_t [getSpaceUsed](#) () const throw (Error::StrategyError)
Obtain real storage utilization.
- void [sync](#) () const throw (Error::StrategyError)
- 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)
- void [replace](#) (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t [length](#) (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void [flush](#) (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t [sequence](#) (string &key, void *const data=NULL, int cursor=[BE_RECSTORE_SEQ_NEXT](#)) throw (Error::ObjectDoesNotExist, Error::StrategyError)
Sequence through a [RecordStore](#), returning the key/data pairs.
- void [setCursorAtKey](#) (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Protected Member Functions

- void [sqliteError](#) (int32_t errorNumber) const throw (Error::StrategyError)
Convert an SQLite error into a StrategyError.
- void [createStructure](#) () throw (Error::StrategyError)
Create the tables needed to store key->value pairs in SQLite.
- bool [validateKeyValueTable](#) (const string &table) throw (Error::StrategyError)
Confirm that a key->value table exists with the proper schema.
- void [createKeyValueTable](#) (const string &table) throw (Error::StrategyError)
Create a tables needed to store key->value pairs in SQLite.
- bool [validateSchema](#) () throw (Error::StrategyError)
Confirm that the schema of the opened SQLite database is compatible.
- uint64_t [readSegments](#) (const string &key, void *const data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
Select a row from the [RecordStore](#).
- void [cleanup](#) () throw (Error::StrategyError)
Perform SQLite cleanup routines.

E.83.1 Detailed Description

A [RecordStore](#) implementation using a SQLite database as the underlying record storage system.

E.83.2 Member Function Documentation

E.83.2.1 void BiometricEvaluation::IO::SQLiteRecordStore::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 RecordStore .
----	-------------	--

Exceptions

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

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.83.2.2 void BiometricEvaluation::IO::SQLiteRecordStore::changeDescription (const string & *description*) throw (Error::StrategyError) [virtual]

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

Parameters

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

Exceptions

<i>Error::StrategyError</i>	An error occurred when using the underlying storage system.
---	---

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.83.2.3 `uint64_t BiometricEvaluation::IO::SQLiteRecordStore::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

<i>Error::StrategyError</i>	An error occurred when using the underlying storage system.
---	---

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.83.2.4 `void BiometricEvaluation::IO::SQLiteRecordStore::sync () const throw (Error::StrategyError) [virtual]`

Synchronize the entire record store to persistent storage.

Exceptions

<i>Error::StrategyError</i>	An error occurred when using the underlying storage system.
---	---

Reimplemented from [BiometricEvaluation::IO::RecordStore](#).

E.83.2.5 `void BiometricEvaluation::IO::SQLiteRecordStore::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

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

Exceptions

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

Implements [BiometricEvaluation::IO::RecordStore](#).

E.83.2.6 `void BiometricEvaluation::IO::SQLiteRecordStore::remove (const string & key) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Remove a record from the store.

Parameters

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

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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.83.2.7 `uint64_t BiometricEvaluation::IO::SQLiteRecordStore::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	<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

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.83.2.8 `void BiometricEvaluation::IO::SQLiteRecordStore::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	<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.83.2.9 `uint64_t BiometricEvaluation::IO::SQLiteRecordStore::length (const string & key) const throw (Error::ObjectDoesNotExist, Error::StrategyError) [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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.83.2.10 `void BiometricEvaluation::IO::SQLiteRecordStore::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

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.83.2.11 `uint64_t BiometricEvaluation::IO::SQLiteRecordStore::sequence (string & key, void *const data = NULL, int cursor = BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

Sequence through a [RecordStore](#), returning the key/data pairs.

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

Parameters

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

Returns

The length of the record currently in sequence.

Exceptions

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.83.2.12 `void BiometricEvaluation::IO::SQLiteRecordStore::setCursorAtKey (string & key) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]`

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

Parameters

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

Exceptions

<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.

Implements [BiometricEvaluation::IO::RecordStore](#).

E.83.2.13 void [BiometricEvaluation::IO::SQLiteRecordStore::sqliteError](#) (int32_t *errorNumber*) const throw (Error::StrategyError) [protected]

Convert an SQLite error into a StrategyError.

Exceptions

<i>Error::StrategyError</i>	Always thrown with the textual description of the last error condition.
---	---

E.83.2.14 void [BiometricEvaluation::IO::SQLiteRecordStore::createStructure](#) () throw (Error::StrategyError) [protected]

Create the tables needed to store key->value pairs in SQLite.

Exceptions

<i>Error::StrategyError</i>	Error executing SQL commands.
---	-------------------------------

E.83.2.15 bool [BiometricEvaluation::IO::SQLiteRecordStore::validateKeyValueTable](#) (const string & *table*) throw (Error::StrategyError) [protected]

Confirm that a key->value table exists with the proper schema.

Parameters

<i>table</i>	Name of the table to check.
--------------	-----------------------------

Returns

Whether or not the table exists with the proper schema.

Exceptions

<i>Error::StrategyError</i>	Error compiling SQL.
---	----------------------

E.83.2.16 void BiometricEvaluation::IO::SQLiteRecordStore::createKeyValueTable (const string & *table*)
throw (Error::StrategyError) [protected]

Create a tables needed to store key->value pairs in SQLite.

Parameters

<i>table</i>	Name of the table to create.
--------------	------------------------------

Exceptions

Error::StrategyError	Error executing SQL commands.
--------------------------------------	-------------------------------

E.83.2.17 bool BiometricEvaluation::IO::SQLiteRecordStore::validateSchema () throw
(Error::StrategyError) [protected]

Confirm that the schema of the opened SQLite database is compatible.

Returns

Whether or not the schema of the opened SQLite database is compatible with this object.

Exceptions

Error::StrategyError	Error compiling SQL.
--------------------------------------	----------------------

E.83.2.18 uint64_t BiometricEvaluation::IO::SQLiteRecordStore::readSegments (const string & *key*,
void *const *data*) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
[protected]

Select a row from the [RecordStore](#).

Parameters

<i>key</i>	Key of the row to select.
<i>data</i>	If not NULL, deep copy the record for key into data.

Exceptions

Error::ObjectDoesNotExist	Key does not exist in RecordStore .
Error::StrategyError	Error executing SQL commands.

Returns

Size of key's record.

E.83.2.19 `void BiometricEvaluation::IO::SQLiteRecordStore::cleanup () throw (Error::StrategyError)`
 [protected]

Perform SQLite cleanup routines.

- Finalize the sequencer statement
- Close the SQLite database handle

Exceptions

<i>Error::StrategyError</i>	Bad return code from SQLite during cleanup.
---	---

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

- `be_io_sqliterecstore.h`

E.84 BiometricEvaluation::Process::Statistics Class Reference

The [Statistics](#) class provides an interface for gathering process statistics, such as memory usage, system time, etc.

```
#include <be_process_statistics.h>
```

Public Member Functions

- [Statistics](#) ()
- [Statistics](#) ([IO::LogCabinet](#) *const logCabinet) throw (Error::NotImplemented, Error::ObjectExists, Error::StrategyError)
- void [getCPUTimes](#) (uint64_t *usertime, uint64_t *systemtime) throw (Error::StrategyError, Error::NotImplemented)
- void [getMemorySizes](#) (uint64_t *vmrss, uint64_t *vmsize, uint64_t *vmpeak, uint64_t *vmdata, uint64_t *vmstack) throw (Error::StrategyError, Error::NotImplemented)
- uint32_t [getNumThreads](#) () throw (Error::StrategyError, Error::NotImplemented)
- void [logStats](#) () throw (Error::ObjectDoesNotExist, Error::StrategyError, Error::NotImplemented)

Create a snapshot of the current process statistics in the LogSheet created in the LogCabinet.

- void [startAutoLogging](#) (uint64_t interval) throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError, Error::NotImplemented)

Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.

- void [stopAutoLogging](#) () throw (Error::ObjectDoesNotExist, Error::StrategyError)

Stop the automatic logging of process statistics.

- void [callStatistics_logStats](#) ()

E.84.1 Detailed Description

The [Statistics](#) class provides an interface for gathering process statistics, such as memory usage, system time, etc.

The information gathered by objects of this class are for the current process, and can optionally be logged to a LogSheet object contained within the provided LogCabinet.

Note

The resolution of a returned value for many methods may not match the resolution allowed by the interface. For example, the operating system may allow for second resolution whereas the interface allows microsecond resolution.

E.84.2 Constructor & Destructor Documentation

E.84.2.1 BiometricEvaluation::Process::Statistics::Statistics ()

Constructor with no parameters.

E.84.2.2 BiometricEvaluation::Process::Statistics::Statistics (IO::LogCabinet *const *logCabinet*) throw (Error::NotImplemented, Error::ObjectExists, Error::StrategyError)

Construct a [Statistics](#) object with the associated LogCabinet.

Parameters

<i>in</i>	<i>logCabinet</i>	The LogCabinet object where this object will create a LogSheet to contain the statistic information for the process.
-----------	-------------------	--

Exceptions

Error::NotImplemented	Logging is not supported on this OS. This exception can be thrown when any portion of the statistics gathering cannot be completed.
Error::ObjectExists	The LogSheet already exists. This exception should rarely, if ever, occur.
Error::StrategyError	Failure to create the LogSheet in the cabinet.

E.84.3 Member Function Documentation

E.84.3.1 void BiometricEvaluation::Process::Statistics::getCPUTimes (uint64_t * *usertime*, uint64_t * *systemtime*) throw (Error::StrategyError, Error::NotImplemented)

Obtain the total user and system times for the process, in microseconds. Any of the out parameters can be NULL, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

out	<i>usertime</i>	Pointer where to store the total user time.
out	<i>systemtime</i>	Pointer where to store the total system time.

Exceptions

<i>Error::StrategyError</i>	An error occurred when obtaining the process statistics from the operating system. The exception information string contains the error reason.
<i>Error::NotImplemented</i>	This method is not implemented on this OS.

E.84.3.2 void BiometricEvaluation::Process::Statistics::getMemorySizes (uint64_t * *vmrss*, uint64_t * *vmsize*, uint64_t * *vmpeak*, uint64_t * *vmdata*, uint64_t * *vmstack*) throw (Error::StrategyError, Error::NotImplemented)

Obtain the current memory set sizes for the process, in kilobytes. Any of the out parameters can be NULL, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

out	<i>vmrss</i>	Pointer where to store the current resident set size.
out	<i>vmsize</i>	Pointer where to store the current total virtual memory size.
out	<i>vmpeak</i>	Pointer where to store the peak total virtual memory size.
out	<i>vmdata</i>	Pointer where to store the current virtual memory data segment size.
out	<i>vmstack</i>	Pointer where to store the current virtual memory stack segment size.

Exceptions

<i>Error::StrategyError</i>	An error occurred when obtaining the process statistics from the operating system. The exception information string contains the error reason.
<i>Error::NotImplemented</i>	This method is not implemented on this OS.

E.84.3.3 uint32_t BiometricEvaluation::Process::Statistics::getNumThreads () throw (Error::StrategyError, Error::NotImplemented)

Obtain the number of threads composing this process.

Note

This method may not be implemented in all operating systems.

Exceptions

<i>Error::StrategyError</i>	An error occurred when obtaining the process info from the operating system. The exception information string contains the error reason.
<i>Error::NotImplemented</i>	This method is not implemented on this OS.

E.84.3.4 void BiometricEvaluation::Process::Statistics::logStats () throw (Error::ObjectDoesNotExist, Error::StrategyError, Error::NotImplemented)

Create a snapshot of the current process statistics in the LogSheet created in the LogCabinet.

Exceptions

<i>Error::ObjectDoesNotExist</i>	The LogSheet does not exist; this object was not created with LogCabinet object.
<i>Error::StrategyError</i>	An error occurred when writing to the LogSheet.
<i>Error::NotImplemented</i>	The statistics gathering is not implemented for this operating system.

E.84.3.5 void BiometricEvaluation::Process::Statistics::startAutoLogging (uint64_t interval) throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError, Error::NotImplemented)

Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.

Note

It is unrealistic to expect that log entries can be made at a rate of one per microsecond. If [stopAutoLogging\(\)](#) is called very soon after the start, a log entry may not be made.

Parameters

in	<i>interval</i>	The gap between logging snapshots, in microseconds.
----	-----------------	---

Exceptions

<i>Error::ObjectDoesNotExist</i>	The LogSheet does not exist; this object was not created with LogCabinet object.
<i>Error::ObjectExists</i>	Autologging is currently invoked.
<i>Error::StrategyError</i>	An error occurred when writing to the LogSheet.
<i>Error::NotImplemented</i>	The statistics gathering is not implemented for this operating system.

E.84.3.6 void BiometricEvaluation::Process::Statistics::stopAutoLogging () throw (Error::ObjectDoesNotExist, Error::StrategyError)

Stop the automatic logging of process statistics.

Exceptions

<i>Error::ObjectDoesNotExist</i>	Not currently autologging.
<i>Error::StrategyError</i>	An error occurred when stopping, most likely because the logging thread died.

E.84.3.7 void BiometricEvaluation::Process::Statistics::callStatistics_logStats ()

Helper function in C++ space that has access to this object, and is called from C space by the logging thread. Applications should not call this function.

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

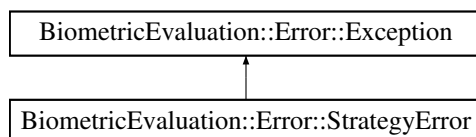
- be_process_statistics.h

E.85 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.85.1 Detailed Description

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

E.85.2 Constructor & Destructor Documentation

E.85.2.1 BiometricEvaluation::Error::StrategyError::StrategyError ()

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

Returns

The [StrategyError](#) object.

E.85.2.2 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

E.86 BiometricEvaluation::Time::Timer Class Reference

This class can be used by applications to report the amount of time a block of code takes to execute.

```
#include <be_time_timer.h>
```

Public Member Functions

- [Timer](#) ()
- void [start](#) () throw (Error::StrategyError)
- void [stop](#) () throw (Error::StrategyError)
- uint64_t [elapsed](#) () throw (Error::StrategyError)

E.86.1 Detailed Description

This class can be used by applications to report the amount of time a block of code takes to execute.

Applications wrap the block of code in the [Timer::start\(\)](#) and [Timer::stop\(\)](#) calls, then use [Timer::elapsed\(\)](#) to obtain the calculated time of the operation.

E.86.2 Constructor & Destructor Documentation

E.86.2.1 BiometricEvaluation::Time::Timer::Timer ()

Constructor for the [Timer](#) object.

E.86.3 Member Function Documentation

E.86.3.1 void BiometricEvaluation::Time::Timer::start () throw (Error::StrategyError)

Start tracking time.

Exceptions

Error::StrategyError	This object is currently timing an operation or an error occurred when obtaining timing information.
--------------------------------------	--

E.86.3.2 void BiometricEvaluation::Time::Timer::stop () throw (Error::StrategyError)

Stop tracking time.

Exceptions

<i>Error::StrategyError</i>	This object is not currently timing an operation or an error occurred when obtaining timing information.
---	--

E.86.3.3 uint64_t BiometricEvaluation::Time::Timer::elapsed () throw (Error::StrategyError)

Get the elapsed time in microseconds between calls to this object's [start\(\)](#) and [stop\(\)](#) methods.

Returns

The number of microseconds between calls to this object's [start\(\)](#) and [stop\(\)](#) methods.

Exceptions

<i>Error::StrategyError</i>	This object is currently timing an operation or an error occurred when obtaining timing information.
---	--

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

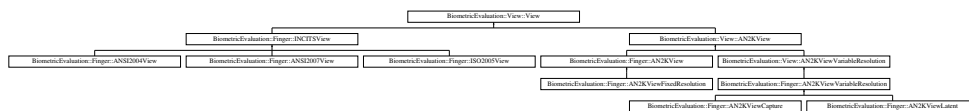
- `be_time_timer.h`

E.87 BiometricEvaluation::View::View Class Reference

A class to represent single biometric element view.

```
#include <be_view_view.h>
```

Inheritance diagram for BiometricEvaluation::View::View:



Public Member Functions

- virtual `tr1::shared_ptr < Image::Image > getImage () const =0`
Obtain the image used for the finger view.
- virtual `Image::Size getImageSize () const =0`
Obtain the image size.
- virtual `Image::Resolution getImageResolution () const =0`
Obtain the image resolution.

- virtual uint32_t [getImageDepth](#) () const =0
Obtain the image depth.
- virtual Image::CompressionAlgorithm::Kind [getCompressionAlgorithm](#) () const =0
Obtain the compression algorithm used on the image.
- virtual Image::Resolution [getScanResolution](#) () const =0
Obtain the image scan resolution.

E.87.1 Detailed Description

A class to represent single biometric element view.

Included in a view is the biometric image and any derived information, such as minutiae points.

E.87.2 Member Function Documentation

E.87.2.1 virtual tr1::shared_ptr<Image::Image> BiometricEvaluation::View::View::getImage () const [pure virtual]

Obtain the image used for the finger view.

Not all finger views will have an image, however the derived information, such as minutiae, may be present.

Implemented in [BiometricEvaluation::View::AN2KView](#), and [BiometricEvaluation::Finger::INCITSView](#).

E.87.2.2 virtual Image::Size BiometricEvaluation::View::View::getImageSize () const [pure virtual]

Obtain the image size.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image size must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

Implemented in [BiometricEvaluation::View::AN2KView](#), and [BiometricEvaluation::Finger::INCITSView](#).

E.87.2.3 virtual Image::Resolution BiometricEvaluation::View::View::getImageResolution () const [pure virtual]

Obtain the image resolution.

[Image](#) resolution is taken from the biometric record, and not from the image data. In some cases, the resolution may be the components of the pixel ratio, and applications must check the [Image::Resolution::units](#) field for value NA.

Implemented in [BiometricEvaluation::View::AN2KView](#), and [BiometricEvaluation::Finger::INCITSView](#).

E.87.2.4 virtual uint32_t BiometricEvaluation::View::View::getImageDepth () const [pure virtual]

Obtain the image depth.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image depth must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

Implemented in [BiometricEvaluation::View::AN2KView](#), and [BiometricEvaluation::Finger::INCITSView](#).

E.87.2.5 `virtual Image::CompressionAlgorithm::Kind BiometricEvaluation::View::View::getCompressionAlgorithm () const` [pure virtual]

Obtain the compression algorithm used on the image.

This value is as present in the biometric record, and not obtained from the image data itself.

Implemented in [BiometricEvaluation::View::AN2KView](#), and [BiometricEvaluation::Finger::INCITSView](#).

E.87.2.6 `virtual Image::Resolution BiometricEvaluation::View::View::getScanResolution () const` [pure virtual]

Obtain the image scan resolution.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image resolution must be equal, but applications can check for inconsistencies.

Implemented in [BiometricEvaluation::View::AN2KView](#), and [BiometricEvaluation::Finger::INCITSView](#).

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

- `be_view_view.h`

E.88 BiometricEvaluation::Time::Watchdog Class Reference

A [Watchdog](#) object can be used by applications to limit the amount of processing time taken by a block of code.

```
#include <be_time_watchdog.h>
```

Public Member Functions

- [Watchdog](#) (const uint8_t type) throw (Error::NotImplemented, Error::ParameterError)
- void [setInterval](#) (uint64_t interval)
- void [start](#) () throw (Error::StrategyError)
- void [stop](#) () throw (Error::StrategyError)
- bool [expired](#) ()
- void [setCanSigJump](#) ()
- void [clearCanSigJump](#) ()
- void [setExpired](#) ()
- void [clearExpired](#) ()

Static Public Attributes

- static const uint8_t [PROCESSTIME](#) = 0

- static const uint8_t [REALTIME](#) = 1
- static bool [_canSigJump](#)
- static sigjmp_buf [_sigJumpBuf](#)

E.88.1 Detailed Description

A [Watchdog](#) object can be used by applications to limit the amount of processing time taken by a block of code.

A [Watchdog](#) object is used to set a timer that, upon expiration, will force a jump to a location within the process. An application can detect whether the timer expired at that point in the code. [Watchdog](#) builds on the POSIX `setitimer(2)` call. [Timer](#) intervals are in terms of process virtual time or real time, based on how the object is constructed.

Most applications will not directly invoke the methods of the `WatchDog` class, instead using the `BEGIN_WATCHDOG_BLOCK()` and `END_WATCHDOG_BLOCK()` macros. Applications should not install their own signal handlers, but use the `SignalManager` class instead.

The `BEGIN_WATCHDOG_BLOCK` macro sets up the jump block and tells the [Watchdog](#) object to start handling the alarm signal. Applications must call [setInterval\(\)](#) before invoking the `BEGIN_WATCHDOG_BLOCK()` macro.

The `END_WATCHDOG_BLOCK()` macro disables the watchdog timer, but doesn't affect the current interval value. Applications can set the interval once and use the `BEGIN/END` block macros repeatedly. Failure to call [setInterval\(\)](#) results in an effectively disabled timer, as does setting the interval to 0.

Note

[Process](#) virtual timing may not be available on all systems. In those cases, an application compilation error will occur because `PROCESSTIME` will not be defined.

Attention

On many systems, the `sleep(3)` call is implemented using alarm signals, the same technique used by the [Watchdog](#) class. Therefore, applications should not call `sleep(3)` inside the [Watchdog](#) block; behavior is undefined in that case, but usually results in cancellation of the [Watchdog](#) timer.

The [setCanSigJump\(\)](#), [clearCanSigJump\(\)](#), [setExpired\(\)](#) and [clearExpired\(\)](#) methods are not meant to be used directly by applications, which should use the `BEGIN_WATCHDOG_BLOCK()/END_WATCHDOG_BLOCK()` macro pair.

See also

[Error::SignalManager](#)

E.88.2 Constructor & Destructor Documentation

E.88.2.1 BiometricEvaluation::Time::Watchdog::Watchdog (const uint8_t *type*) throw (Error::NotImplemented, Error::ParameterError)

Construct a new [Watchdog](#) object.

Parameters

<i>in</i>	<i>type</i>	The type of timer, <code>ProcessTime</code> or <code>RealTime</code> .
-----------	-------------	--

Returns

The [Watchdog](#) object.

Exceptions

Error::NotImplemented	The type of watchdog requested is not implemented.
Error::ParameterError	The type is invalid.

Warning

[Watchdog::PROCESSTIME](#) is not supported under Cygwin.

E.88.3 Member Function Documentation**E.88.3.1 void BiometricEvaluation::Time::Watchdog::setInterval (uint64_t *interval*)**

Set the interval for the timer, but don't start the timer. Setting a value of 0 will essentially disable the timer. [Timer](#) intervals are in microseconds, however actual intervals are dependent on the resolution of the system clock, and may not be at microsecond resolution.

Parameters

<i>in</i>	<i>interval</i>	The timer interval, in microseconds.
-----------	-----------------	--------------------------------------

E.88.3.2 void BiometricEvaluation::Time::Watchdog::start () throw (Error::StrategyError)

Start a watchdog timer.

Exceptions

Error::StrategyError	Could not register the signal handler, or could not create the timer.
--------------------------------------	---

E.88.3.3 void BiometricEvaluation::Time::Watchdog::stop () throw (Error::StrategyError)

Stop a watchdog timer.

Exceptions

Error::StrategyError	Could not clear the timer.
--------------------------------------	----------------------------

E.88.3.4 `bool BiometricEvaluation::Time::Watchdog::expired ()`

Indicate whether the watchdog timer expired.

Returns

true if the timer expired, false otherwise.

E.88.3.5 `void BiometricEvaluation::Time::Watchdog::setCanSigJump ()`

Indicate that the signal handler can jump into the application code after handling the signal.

E.88.3.6 `void BiometricEvaluation::Time::Watchdog::clearCanSigJump ()`

Clears the flag for the [Watchdog](#) object to indicate that the signal jump block is no longer valid.

E.88.3.7 `void BiometricEvaluation::Time::Watchdog::setExpired ()`

Set a flag to indicate the timer expired.

E.88.3.8 `void BiometricEvaluation::Time::Watchdog::clearExpired ()`

Clear the flag indicating the timer expired.

E.88.4 Member Data Documentation**E.88.4.1** `const uint8_t BiometricEvaluation::Time::Watchdog::PROCESSTIME = 0` [static]

A [Watchdog](#) based on process time.

E.88.4.2 `const uint8_t BiometricEvaluation::Time::Watchdog::REALTIME = 1` [static]

A [Watchdog](#) based on real (wall clock) time.

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

- `be_time_watchdog.h`

E.89 BiometricEvaluation::Process::Worker Class Reference

An abstraction of an instance that performs work on given data.

```
#include <be_process_worker.h>
```

Public Member Functions

- virtual int32_t [workerMain](#) ()=0
The method that will get called to to start execution by a [ProcessManager](#).
- tr1::shared_ptr< void > [getParameter](#) (const string &name)
Obtain a parameter passed to this [Worker](#).
- double [getParameterAsDouble](#) (const string &name)
Obtain a parameter passed to this [Worker](#) as a double.
- int64_t [getParameterAsInteger](#) (const string &name)
Obtain a parameter passed to this [Worker](#) as an integer.
- string [getParameterAsString](#) (const string &name)
Obtain a parameter passed to this [Worker](#) as a string.
- void [setParameter](#) (const string &name, tr1::shared_ptr< void > argument)
Pass a parameter to this [Worker](#).
- void [stop](#) ()
Tell this [Worker](#) to return ASAP.
- virtual [~Worker](#) ()
[Worker](#) destructor.

Protected Member Functions

- [Worker](#) ()
[Worker](#) constructor.
- bool [stopRequested](#) () const
Determine if the parent has requested this child to exit.

E.89.1 Detailed Description

An abstraction of an instance that performs work on given data.

E.89.2 Member Function Documentation

E.89.2.1 virtual int32_t BiometricEvaluation::Process::Worker::workerMain () [pure virtual]

The method that will get called to to start execution by a [ProcessManager](#).

Returns

Status code.

Note

If an object of this class is added to a [Process::ForkManager](#) object, the implementation of [Process::Worker::workerMain\(\)](#) should release all resources prior to returning.

E.89.2.2 `tr1::shared_ptr<void> BiometricEvaluation::Process::Worker::getParameter (const string & name)`

Obtain a parameter passed to this [Worker](#).

Parameters

<i>name</i>	The parameter name to retrieve.
-------------	---------------------------------

Returns

`shared_ptr` to the parameter argument.

Attention

If name does not exist, a new `shared_ptr` will be set for name.

E.89.2.3 `double BiometricEvaluation::Process::Worker::getParameterAsDouble (const string & name)`

Obtain a parameter passed to this [Worker](#) as a double.

Parameters

<i>name</i>	The parameter name to retrieve.
-------------	---------------------------------

Returns

Parameter as a double.

Attention

If name does not exist, a new `shared_ptr<double>` will be set for name.

E.89.2.4 `int64_t BiometricEvaluation::Process::Worker::getParameterAsInteger (const string & name)`

Obtain a parameter passed to this [Worker](#) as an integer.

Parameters

<i>name</i>	The parameter name to retrieve.
-------------	---------------------------------

Returns

Parameter as an integer.

Attention

If name does not exist, a new `shared_ptr<int64_t>` will be set for name.

E.89.2.5 `string BiometricEvaluation::Process::Worker::getParameterAsString (const string & name)`

Obtain a parameter passed to this [Worker](#) as a string.

Parameters

<i>name</i>	The parameter name to retrieve.
-------------	---------------------------------

Returns

Parameter as a string.

Attention

If name does not exist, a new `shared_ptr<string>` will be set for name.

E.89.2.6 `void BiometricEvaluation::Process::Worker::setParameter (const string & name, tr1::shared_ptr<void > argument)`

Pass a parameter to this [Worker](#).

Parameters

<i>name</i>	A unique identifier for this parameter
<i>argument</i>	A <code>shared_ptr</code> to the object to store.

E.89.2.7 `void BiometricEvaluation::Process::Worker::stop ()`

Tell this [Worker](#) to return ASAP.

Attention

This method should not be overridden.

E.89.2.8 `bool BiometricEvaluation::Process::Worker::stopRequested () const` `[protected]`

Determine if the parent has requested this child to exit.

Returns

Whether or not this child should exit.

Attention

This method should not be overridden.

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

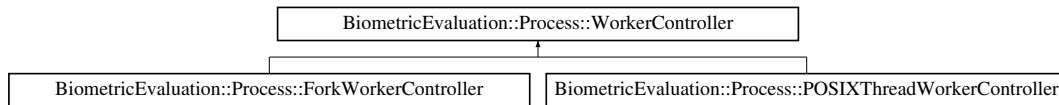
- `be_process_worker.h`

E.90 BiometricEvaluation::Process::WorkerController Class Reference

Wrapper of a [Worker](#) returned from a [Process::Manager](#).

```
#include <be_process_workercontroller.h>
```

Inheritance diagram for BiometricEvaluation::Process::WorkerController:



Public Member Functions

- [WorkerController](#) (tr1::shared_ptr< [Worker](#) > worker)
- virtual void [setParameter](#) (const string &name, tr1::shared_ptr< void > argument)
Set the parameter to be passed to the [Worker](#).
- virtual void [setParameterFromDouble](#) (const string &name, double argument)
Set a double parameter to be passed to the [Worker](#).
- virtual void [setParameterFromInteger](#) (const string &name, int64_t argument)
Set an integer parameter to be passed to the [Worker](#).
- virtual void [setParameterFromString](#) (const string &name, const string &argument)
Set a string parameter to be passed to the [Worker](#).
- virtual void [reset](#) () throw (Error::ObjectExists)
Reuse the [Worker](#).
- virtual bool [isWorking](#) () const =0
Obtain whether or not [Worker](#) is working.
- tr1::shared_ptr< [Worker](#) > [getWorker](#) () const
Obtain the [Worker](#) instance being wrapped.
- virtual [~WorkerController](#) ()
[WorkerController](#) destructor.

Protected Attributes

- tr1::shared_ptr< [Worker](#) > [_worker](#)

E.90.1 Detailed Description

Wrapper of a [Worker](#) returned from a [Process::Manager](#).

E.90.2 Constructor & Destructor Documentation

E.90.2.1 BiometricEvaluation::Process::WorkerController::WorkerController (tr1::shared_ptr< [Worker](#) > worker)

[WorkerController](#) constructor.

Parameters

<i>worker</i>	The Worker instance to wrap.
---------------	--

E.90.3 Member Function Documentation

E.90.3.1 `virtual void BiometricEvaluation::Process::WorkerController::setParameter (const string & name, tr1::shared_ptr< void > argument) [virtual]`

Set the parameter to be passed to the [Worker](#).

Parameters

<i>in</i>	<i>name</i>	The name representing the argument in the Worker .
<i>in</i>	<i>argument</i>	The argument to be passed to the Worker .

Note

Subsequent calls to [setParameter\(\)](#) with the same name will overwrite any exiting argument.

E.90.3.2 `virtual void BiometricEvaluation::Process::WorkerController::setParameterFromDouble (const string & name, double argument) [virtual]`

Set a double parameter to be passed to the [Worker](#).

Parameters

<i>in</i>	<i>name</i>	The name representing the argument in the Worker .
<i>in</i>	<i>argument</i>	The double to be passed to the Worker .

Note

Subsequent calls to [setParameter*\(\)](#) with the same name will overwrite any exiting argument.

E.90.3.3 `virtual void BiometricEvaluation::Process::WorkerController::setParameterFromInteger (const string & name, int64_t argument) [virtual]`

Set an integer parameter to be passed to the [Worker](#).

Parameters

<i>in</i>	<i>name</i>	The name representing the argument in the Worker .
<i>in</i>	<i>argument</i>	The integer to be passed to the Worker .

Note

Subsequent calls to [setParameter*\(\)](#) with the same name will overwrite any exiting argument.

E.90.3.4 `virtual void BiometricEvaluation::Process::WorkerController::setParameterFromString (const string & name, const string & argument) [virtual]`

Set a string parameter to be passed to the [Worker](#).

Parameters

in	<i>name</i>	The name representing the argument in the Worker .
in	<i>argument</i>	The string to be passed to the Worker .

Note

Subsequent calls to `setParameter*()` with the same name will overwrite any exiting argument.

E.90.3.5 `virtual void BiometricEvaluation::Process::WorkerController::reset () throw (Error::ObjectExists) [virtual]`

Reuse the [Worker](#).

Exceptions

Error::ObjectExists	The previously started Worker is still running.
-------------------------------------	---

Reimplemented in [BiometricEvaluation::Process::ForkWorkerController](#), and [BiometricEvaluation::Process::PO-
SIXThreadWorkerController](#).

E.90.3.6 `virtual bool BiometricEvaluation::Process::WorkerController::isWorking () const [pure
virtual]`

Obtain whether or not [Worker](#) is working.

Returns

Whether or not the [Worker](#) is working.

Implemented in [BiometricEvaluation::Process::ForkWorkerController](#), and [BiometricEvaluation::Process::PO-
SIXThreadWorkerController](#).

E.90.3.7 `tr1::shared_ptr<Worker> BiometricEvaluation::Process::WorkerController::getWorker () const`

Obtain the [Worker](#) instance being wrapped.

Returns

[Worker](#) instance.

E.90.4 Member Data Documentation

E.90.4.1 `tr1::shared_ptr<Worker> BiometricEvaluation::Process::WorkerController::_worker`
`[protected]`

The [Worker](#) instance that is running in this child

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

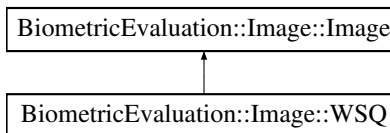
- `be_process_workercontroller.h`

E.91 BiometricEvaluation::Image::WSQ Class Reference

A WSQ-encoded image.

```
#include <be_image_wsq.h>
```

Inheritance diagram for `BiometricEvaluation::Image::WSQ`:



Public Member Functions

- **WSQ** (`const uint8_t *data, const uint64_t size`) `throw (Error::DataError, Error::StrategyError)`
- `Memory::AutoArray< uint8_t > getRawData ()` `const throw (Error::DataError)`
Accessor for the raw image data. The data returned should not be compressed or encoded.
- `Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8)` `const throw (Error::DataError, Error::ParameterError)`
Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool `isWSQ (const uint8_t *data)`

E.91.1 Detailed Description

A WSQ-encoded image.

E.91.2 Member Function Documentation

E.91.2.1 `Memory::AutoArray<uint8_t> BiometricEvaluation::Image::WSQ::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

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

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

E.91.2.2 `Memory::AutoArray<uint8_t> BiometricEvaluation::Image::WSQ::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

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

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. *depth* adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

E.91.2.3 `static bool BiometricEvaluation::Image::WSQ::isWSQ (const uint8_t * data) [static]`

Whether or not data is a [WSQ](#) image.

Parameters

<i>in</i>	<i>data</i>	The buffer to check.
-----------	-------------	----------------------

Returns

true if data appears to be a [WSQ](#) image, false otherwise

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

- `be_image_wsq.h`