

# BIOMETRIC EVALUATION COMMON FRAMEWORK

PROGRAMMER'S GUIDE

VERSION 0.1

---

WAYNE SALAMON  
GREGORY FIUMARA

IMAGE GROUP  
INFORMATION ACCESS DIVISION  
INFORMATION TECHNOLOGY LABORATORY



JANUARY 15, 2015

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Overview</b>	<b>3</b>
<b>3</b>	<b>Framework</b>	<b>5</b>
3.1	Versioning . . . . .	5
3.2	Enumerations . . . . .	5
<b>4</b>	<b>Memory</b>	<b>7</b>
4.1	AutoBuffer . . . . .	7
4.2	AutoArray . . . . .	8
4.3	IndexedBuffer . . . . .	9
<b>5</b>	<b>Error Handling</b>	<b>11</b>
5.1	Biometric Evaluation Exceptions . . . . .	11
5.2	Signal Handling . . . . .	11
<b>6</b>	<b>Input/Output</b>	<b>15</b>
6.1	Utility . . . . .	15
6.2	Record Management . . . . .	15
6.3	Logging . . . . .	16
6.3.1	FileLogsheet . . . . .	17
6.3.2	SysLogsheet . . . . .	18
6.4	Properties . . . . .	18
6.5	Compressor . . . . .	19
<b>7</b>	<b>Time and Timing</b>	<b>21</b>
7.1	Elapsed Time . . . . .	21
7.2	Limiting Execution Time . . . . .	21
<b>8</b>	<b>Process Information and Control</b>	<b>23</b>
8.1	Process Statistics . . . . .	23
8.2	Process Management . . . . .	25
8.2.1	Manager . . . . .	25
8.2.2	Worker . . . . .	25
8.2.3	WorkerController . . . . .	26
8.2.4	Communications . . . . .	28
<b>9</b>	<b>System</b>	<b>31</b>

<b>10 Image</b>	<b>33</b>
10.1 The Image Namespace	33
10.2 The Image Class	33
10.3 Raw Image	34
10.4 JPEG	34
10.5 JPEGL	34
10.6 JPEG2000	34
10.7 NetPBM	35
10.8 PNG	35
10.9 WSQ	35
<b>11 Text</b>	<b>37</b>
<b>12 Feature</b>	<b>39</b>
12.1 ANSI/NIST Features	39
12.2 ISO/INCITS Features	39
<b>13 Finger</b>	<b>41</b>
13.1 ANSI/NIST Minutiae Data Record	41
13.1.1 ANSI/NIST Finger Views	41
13.1.2 ISO/INCITS Finger Views	43
<b>14 View</b>	<b>45</b>
<b>15 Data Interchange</b>	<b>47</b>
15.1 ANSI/NIST Data Records	47
15.2 INCITS Data Records	50
15.2.1 Finger Views	50
<b>16 Messaging</b>	<b>51</b>
16.1 Message Center	51
16.2 Command Center	52
<b>17 Parallel Processing</b>	<b>55</b>
17.1 MPI Parallel Processing Package	55
17.2 Work Package	55
17.3 Distributor	57
17.3.1 Record Store Distributor	57
17.4 Receiver	57
17.5 Work Package Processor	57
17.5.1 Record Processor	58
17.6 MPI Resources	58
17.7 MPI Runtime	58
17.8 Logging	59
17.9 MPI Framework Applications	59
<b>References</b>	<b>65</b>
<b>A API Reference</b>	<b>67</b>
<b>B Namespace Index</b>	<b>69</b>
B.1 Namespace List	69

<b>C</b>	<b>Hierarchical Index</b>	<b>71</b>
C.1	Class Hierarchy . . . . .	71
<b>D</b>	<b>Class Index</b>	<b>75</b>
D.1	Class List . . . . .	75
<b>E</b>	<b>Namespace Documentation</b>	<b>81</b>
E.1	BiometricEvaluation Namespace Reference . . . . .	81
E.1.1	Detailed Description . . . . .	82
E.2	BiometricEvaluation::Error Namespace Reference . . . . .	82
E.2.1	Detailed Description . . . . .	83
E.2.2	Function Documentation . . . . .	83
	errorStr . . . . .	83
E.3	BiometricEvaluation::Face Namespace Reference . . . . .	83
E.3.1	Detailed Description . . . . .	84
E.3.2	Typedef Documentation . . . . .	84
	PropertySet . . . . .	84
E.4	BiometricEvaluation::Feature Namespace Reference . . . . .	84
E.4.1	Detailed Description . . . . .	86
E.5	BiometricEvaluation::Finger Namespace Reference . . . . .	86
E.5.1	Detailed Description . . . . .	87
E.5.2	Enumeration Type Documentation . . . . .	87
	FingerImageCode . . . . .	87
	Impression . . . . .	88
	PatternClassification . . . . .	88
	Position . . . . .	88
E.5.3	Function Documentation . . . . .	88
	operator<< . . . . .	88
E.6	BiometricEvaluation::Framework Namespace Reference . . . . .	88
E.6.1	Detailed Description . . . . .	90
E.6.2	Function Documentation . . . . .	90
	getCompileDate . . . . .	90
	getCompiler . . . . .	90
	getCompilerVersion . . . . .	90
	getCompileTime . . . . .	90
	getMajorVersion . . . . .	90
	getMinorVersion . . . . .	90
	operator"!= . . . . .	90
	operator"!= . . . . .	91
	operator"!= . . . . .	91
	operator"!= . . . . .	91
	operator+ . . . . .	92
	operator+ . . . . .	92
	operator+ . . . . .	92
	operator+ . . . . .	92
	operator<< . . . . .	93
	operator<< . . . . .	93
	operator== . . . . .	93
	operator== . . . . .	93
	operator== . . . . .	94
	operator== . . . . .	94
E.7	BiometricEvaluation::Image Namespace Reference . . . . .	94

E.7.1	Detailed Description	96
E.7.2	Enumeration Type Documentation	96
	CompressionAlgorithm	96
E.7.3	Function Documentation	96
	distance	96
	operator<<	96
E.8	BiometricEvaluation::IO Namespace Reference	96
E.8.1	Detailed Description	97
E.8.2	Typedef Documentation	98
	ManifestMap	98
E.9	BiometricEvaluation::IO::Utility Namespace Reference	98
E.9.1	Detailed Description	98
E.9.2	Function Documentation	99
	copyDirectoryContents	99
	createTemporaryFile	100
	createTemporaryFile	100
	fileExists	101
	getFileSize	101
	isReadable	101
	isWritable	102
	makePath	102
	readFile	102
	removeDirectory	103
	removeDirectory	103
	setAsideName	103
	sumDirectoryUsage	104
	writeFile	104
	writeFile	104
E.10	BiometricEvaluation::Iris Namespace Reference	105
E.10.1	Detailed Description	105
E.11	BiometricEvaluation::Memory Namespace Reference	105
E.11.1	Detailed Description	106
E.12	BiometricEvaluation::Memory::AutoArrayUtility Namespace Reference	106
E.12.1	Detailed Description	107
E.12.2	Function Documentation	107
	ctr	107
	getString	107
	setString	107
	setString	107
E.13	BiometricEvaluation::MPI Namespace Reference	108
E.13.1	Detailed Description	109
E.13.2	Function Documentation	109
	generateUniqueID	109
	logEntry	109
	logMessage	109
	openLogsheet	109
	printStatus	110
E.14	BiometricEvaluation::Process Namespace Reference	110
E.14.1	Detailed Description	111
E.14.2	Typedef Documentation	111
	ParameterList	111

E.15	BiometricEvaluation::System Namespace Reference	111
E.15.1	Detailed Description	111
E.15.2	Function Documentation	111
	getCPUCount	111
	getLoadAverage	112
	getRealMemorySize	112
E.16	BiometricEvaluation::Text Namespace Reference	112
E.16.1	Detailed Description	113
E.16.2	Function Documentation	113
	basename	113
	caseInsensitiveCompare	113
	digest	113
	digest	114
	dirname	115
	split	115
E.17	BiometricEvaluation::Time Namespace Reference	115
E.17.1	Detailed Description	116
E.17.2	Function Documentation	116
	getCurrentCalendarInformation	116
	getCurrentDate	117
	getCurrentDateAndTime	117
	getCurrentTime	117
	put_time	117
E.18	BiometricEvaluation::View Namespace Reference	117
E.18.1	Detailed Description	117
E.18.2	Function Documentation	118
	operator<<	118
<b>F</b>	<b>Class Documentation</b>	<b>121</b>
F.1	BiometricEvaluation::Feature::AN2K7Minutiae Class Reference	121
F.1.1	Detailed Description	122
F.1.2	Member Enumeration Documentation	122
	EncodingMethod	122
F.1.3	Constructor & Destructor Documentation	122
	AN2K7Minutiae	122
	AN2K7Minutiae	123
F.1.4	Member Function Documentation	123
	convertCoordinate	123
	convertEncodingMethod	123
	convertPatternClassification	124
	convertPatternClassification	124
	getOriginatingFingerprintReadingSystem	124
	getPatternClassificationSet	124
F.2	BiometricEvaluation::Finger::AN2KMinutiaeDataRecord Class Reference	124
F.2.1	Detailed Description	125
F.2.2	Constructor & Destructor Documentation	125
	AN2KMinutiaeDataRecord	125
	AN2KMinutiaeDataRecord	125
F.2.3	Member Function Documentation	126
	getAN2K7Minutiae	126
	getImpressionType	126
	getRegisteredVendorBlock	126

F.3	BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric Struct Reference	126
F.3.1	Detailed Description	127
F.4	BiometricEvaluation::DataInterchange::AN2KRecord Class Reference	127
F.4.1	Detailed Description	128
F.4.2	Member Typedef Documentation	128
	CharacterSet	128
	DomainName	128
F.4.3	Constructor & Destructor Documentation	128
	AN2KRecord	128
	AN2KRecord	129
F.4.4	Member Function Documentation	129
	getDate	129
	getDestinationAgency	129
	getDirectoryOfCharacterSets	129
	getDomainName	129
	getFingerCaptureCount	130
	getFingerCaptures	130
	getFingerLatentCount	130
	getFingerLatents	130
	getGreenwichMeanTime	130
	getMinutiaeDataRecordSet	130
	getNativeScanningResolution	131
	getNominalTransmittingResolution	131
	getOriginatingAgency	131
	getPriority	131
	getTransactionControlNumber	131
	getVersionNumber	131
	recordLocations	131
	recordLocations	132
F.5	BiometricEvaluation::Finger::AN2KView Class Reference	132
F.5.1	Detailed Description	133
F.5.2	Constructor & Destructor Documentation	133
	AN2KView	133
	AN2KView	134
F.5.3	Member Function Documentation	134
	addMinutiaeDataRecord	134
	convertFingerImageCode	134
	convertPosition	134
	getImpressionType	135
	getMinutiaeDataRecordSet	135
	getPositions	135
	populateFGP	135
	setImpressionType	135
	setPositions	135
F.6	BiometricEvaluation::View::AN2KView Class Reference	136
F.6.1	Detailed Description	137
F.6.2	Member Enumeration Documentation	137
	DeviceMonitoringMode	137
	RecordType	137
F.6.3	Constructor & Destructor Documentation	138



	AN2KView . . . . .	138
	AN2KView . . . . .	138
F.6.4	Member Function Documentation . . . . .	138
	convertCompressionAlgorithm . . . . .	138
	convertDeviceMonitoringMode . . . . .	138
	getAN2KRecord . . . . .	138
	getMinutiaeDataRecordSet . . . . .	139
	getRecordType . . . . .	139
F.7	BiometricEvaluation::Finger::AN2KViewCapture Class Reference . . . . .	139
F.7.1	Detailed Description . . . . .	140
F.7.2	Member Enumeration Documentation . . . . .	140
	AmputatedBandaged . . . . .	140
F.7.3	Constructor & Destructor Documentation . . . . .	141
	AN2KViewCapture . . . . .	141
	AN2KViewCapture . . . . .	141
F.7.4	Member Function Documentation . . . . .	141
	convertAlternateFingerSegmentPosition . . . . .	141
	convertAmputatedBandaged . . . . .	141
	convertFingerSegmentPosition . . . . .	142
	extractNISTQuality . . . . .	143
	getAlternateFingerSegmentPositionSet . . . . .	143
	getAmputatedBandaged . . . . .	143
	getFingerprintQualityMetric . . . . .	143
	getFingerSegmentPositionSet . . . . .	143
	getNISTQualityMetric . . . . .	144
	getSegmentationQualityMetric . . . . .	144
F.8	BiometricEvaluation::Finger::AN2KViewFixedResolution Class Reference . . . . .	144
F.8.1	Detailed Description . . . . .	144
F.8.2	Constructor & Destructor Documentation . . . . .	145
	AN2KViewFixedResolution . . . . .	145
	AN2KViewFixedResolution . . . . .	145
F.9	BiometricEvaluation::Finger::AN2KViewLatent Class Reference . . . . .	145
F.9.1	Constructor & Destructor Documentation . . . . .	146
	AN2KViewLatent . . . . .	146
	AN2KViewLatent . . . . .	146
F.9.2	Member Function Documentation . . . . .	146
	getLatentQualityMetric . . . . .	146
F.10	BiometricEvaluation::Finger::AN2KViewVariableResolution Class Reference . . . . .	147
F.10.1	Detailed Description . . . . .	148
F.10.2	Constructor & Destructor Documentation . . . . .	148
	AN2KViewVariableResolution . . . . .	148
	AN2KViewVariableResolution . . . . .	148
F.10.3	Member Function Documentation . . . . .	149
	convertPrintPositionCoordinate . . . . .	149
	getImpressionType . . . . .	150
	getPositionDescriptors . . . . .	150
	getPositions . . . . .	150
	getPrintPositionCoordinates . . . . .	150
	parsePositionDescriptors . . . . .	150
F.11	BiometricEvaluation::View::AN2KViewVariableResolution Class Reference . . . . .	151
F.11.1	Detailed Description . . . . .	152

F.11.2	Constructor & Destructor Documentation	152
	AN2KViewVariableResolution	152
	AN2KViewVariableResolution	152
F.11.3	Member Function Documentation	152
	extractQuality	152
	getCaptureDate	153
	getComment	153
	getQualityMetric	153
	getSourceAgency	153
	getUserDefinedField	153
	parseUserDefinedField	153
F.12	BiometricEvaluation::Finger::ANSI2004View Class Reference	154
F.12.1	Detailed Description	155
F.12.2	Constructor & Destructor Documentation	155
	ANSI2004View	155
	ANSI2004View	155
F.12.3	Member Function Documentation	155
	readCoreDeltaData	155
F.13	BiometricEvaluation::Finger::ANSI2007View Class Reference	156
F.13.1	Detailed Description	156
F.13.2	Constructor & Destructor Documentation	156
	ANSI2007View	156
	ANSI2007View	158
F.13.3	Member Function Documentation	158
	readCoreDeltaData	158
F.14	BiometricEvaluation::IO::ArchiveRecordStore Class Reference	158
F.14.1	Detailed Description	159
F.14.2	Constructor & Destructor Documentation	160
	ArchiveRecordStore	160
	ArchiveRecordStore	160
	~ArchiveRecordStore	160
F.14.3	Member Function Documentation	160
	flush	160
	getArchiveName	161
	getManifestName	161
	getSpaceUsed	161
	insert	161
	length	162
	move	162
	needsVacuum	162
	needsVacuum	162
	read	163
	remove	163
	replace	163
	sequence	164
	setCursorAtKey	164
	sync	165
	vacuum	165
F.14.4	Member Data Documentation	165
	OFFSET_RECORD_REMOVED	165
F.15	BiometricEvaluation::Memory::AutoArray< T > Class Template Reference	165

F.15.1	Detailed Description	167
F.15.2	Member Typedef Documentation	167
	const_iterator	167
	const_reference	167
	iterator	167
	reference	167
	size_type	167
	value_type	167
F.15.3	Constructor & Destructor Documentation	167
	AutoArray	167
	AutoArray	167
	AutoArray	168
	~AutoArray	168
F.15.4	Member Function Documentation	168
	at	168
	at	168
	begin	169
	begin	169
	cbegin	169
	cend	169
	copy	169
	copy	169
	end	170
	end	170
	operator const T *	170
	operator T *	170
	operator=	170
	operator=	171
	operator[]	171
	operator[]	171
	resize	171
	size	172
F.16	BiometricEvaluation::Memory::AutoArrayIterator< B, T > Class Template Reference	172
F.16.1	Detailed Description	173
F.16.2	Member Typedef Documentation	173
	DIFFERENCE	173
F.16.3	Constructor & Destructor Documentation	174
	AutoArrayIterator	174
	AutoArrayIterator	175
	AutoArrayIterator	175
	~AutoArrayIterator	175
F.16.4	Member Function Documentation	175
	operator"!=	175
	operator*	175
	operator+	175
	operator+	175
	operator++	176
	operator++	176
	operator+=	176
	operator-	176
	operator-	176

operator--	176
operator--	176
operator==	176
operator->	177
operator<	177
operator<=	177
operator=	177
operator=	177
operator==	177
operator>	177
operator>=	177
operator[]	178
F.16.5 Friends And Related Function Documentation	178
operator+	178
operator-	178
F.17 BiometricEvaluation::Memory::AutoBuffer< T > Class Template Reference	178
F.17.1 Member Typedef Documentation	178
value_type	178
F.18 BiometricEvaluation::Image::BMP Class Reference	179
F.18.1 Detailed Description	179
F.18.2 Member Function Documentation	180
getRawData	180
getRawGrayscaleData	180
isBMP	180
F.19 BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet Struct Reference	181
F.19.1 Constructor & Destructor Documentation	181
CharacterSet	181
F.19.2 Member Data Documentation	181
commonName	181
identifier	181
version	181
F.20 BiometricEvaluation::Process::CommandCenter< T, typename >::Command Class Reference	181
F.20.1 Detailed Description	182
F.20.2 Member Data Documentation	182
arguments	182
clientId	182
command	182
F.21 BiometricEvaluation::Process::CommandCenter< T, typename > Class Template Reference	182
F.21.1 Detailed Description	183
F.21.2 Constructor & Destructor Documentation	183
CommandCenter	183
~CommandCenter	183
F.21.3 Member Function Documentation	183
disconnectClient	183
getNextCommand	183
hasPendingCommands	184
sendResponse	184
F.22 BiometricEvaluation::Process::CommandParser< T > Class Template Reference	184
F.22.1 Detailed Description	185
F.22.2 Constructor & Destructor Documentation	185
CommandParser	185

~CommandParser	185
F.22.3 Member Function Documentation	185
getNextCommand	185
getUsage	186
parse	186
setUsage	186
F.23 BiometricEvaluation::IO::CompressedRecordStore Class Reference	186
F.23.1 Detailed Description	187
F.23.2 Constructor & Destructor Documentation	187
CompressedRecordStore	187
CompressedRecordStore	187
CompressedRecordStore	188
CompressedRecordStore	188
F.23.3 Member Function Documentation	188
flush	188
getSpaceUsed	189
insert	189
length	189
move	190
operator=	190
read	190
remove	190
replace	191
sequence	191
setCursorAtKey	192
sync	192
F.23.4 Member Data Documentation	192
BACKING_STORE	192
COMPRESSOR_TYPE_KEY	192
F.24 BiometricEvaluation::IO::Compressor Class Reference	192
F.24.1 Detailed Description	194
F.24.2 Member Enumeration Documentation	194
Kind	194
F.24.3 Constructor & Destructor Documentation	194
Compressor	194
~Compressor	194
Compressor	194
F.24.4 Member Function Documentation	194
compress	194
compress	195
compress	195
compress	195
compress	196
compress	196
createCompressor	196
decompress	197
decompress	197
decompress	197
decompress	198
decompress	198
decompress	198

	getOption	199
	getOptionAsInteger	199
	operator=	199
	removeOption	199
	setOption	200
	setOption	200
F.25	BiometricEvaluation::Framework::ConstEnumMapWrapper< T > Class Template Reference	200
F.25.1	Detailed Description	200
F.25.2	Constructor & Destructor Documentation	201
	ConstEnumMapWrapper	201
F.25.3	Member Function Documentation	201
	operator std::string	201
	operator T	201
F.26	BiometricEvaluation::Error::ConversionError Class Reference	201
F.26.1	Detailed Description	201
F.26.2	Constructor & Destructor Documentation	201
	ConversionError	201
	ConversionError	201
F.27	BiometricEvaluation::Image::Coordinate Struct Reference	202
F.27.1	Detailed Description	202
F.27.2	Constructor & Destructor Documentation	202
	Coordinate	202
F.27.3	Member Data Documentation	202
	x	202
	xDistance	202
	y	202
	yDistance	202
F.28	BiometricEvaluation::Feature::CorePoint Struct Reference	203
F.28.1	Detailed Description	203
F.29	BiometricEvaluation::Error::DataError Class Reference	203
F.29.1	Detailed Description	203
F.29.2	Constructor & Destructor Documentation	204
	DataError	204
	DataError	204
F.30	BiometricEvaluation::IO::DBRecordStore Class Reference	204
F.30.1	Detailed Description	204
F.30.2	Constructor & Destructor Documentation	205
	DBRecordStore	205
	DBRecordStore	206
F.30.3	Member Function Documentation	206
	flush	206
	getSpaceUsed	206
	insert	207
	length	207
	move	207
	read	207
	remove	208
	replace	208
	sequence	208
	setCursorAtKey	209
	sync	209

F.31	BiometricEvaluation::Feature::DeltaPoint Struct Reference	209
F.31.1	Detailed Description	210
F.32	BiometricEvaluation::MPI::Distributor Class Reference	210
F.32.1	Detailed Description	210
F.32.2	Constructor & Destructor Documentation	211
	Distributor	211
F.32.3	Member Function Documentation	211
	createWorkPackage	211
	getLogsheet	211
	start	211
F.33	BiometricEvaluation::DataInterchange::AN2KRecord::DomainName Struct Reference	211
F.33.1	Detailed Description	212
F.33.2	Constructor & Destructor Documentation	212
	DomainName	212
F.33.3	Member Data Documentation	212
	identifier	212
	version	212
F.34	BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry Struct Reference	212
F.34.1	Constructor & Destructor Documentation	212
	Entry	212
F.34.2	Member Data Documentation	213
	code	213
	standard	213
F.35	BiometricEvaluation::Framework::EnumerationFunctions< T > Class Template Reference	213
F.35.1	Detailed Description	213
F.35.2	Member Data Documentation	213
	enumToStringMap	213
F.36	BiometricEvaluation::Framework::EnumMapWrapper< T > Class Template Reference	213
F.36.1	Detailed Description	214
F.36.2	Constructor & Destructor Documentation	214
	EnumMapWrapper	214
F.36.3	Member Function Documentation	214
	operator std::string	214
	operator T	214
F.37	BiometricEvaluation::Error::Exception Class Reference	214
F.37.1	Detailed Description	215
F.37.2	Constructor & Destructor Documentation	215
	Exception	215
	Exception	215
F.37.3	Member Function Documentation	216
	what	216
	whatString	216
F.38	BiometricEvaluation::Error::FileError Class Reference	216
F.38.1	Detailed Description	216
F.38.2	Constructor & Destructor Documentation	216
	FileError	216
	FileError	217
F.39	BiometricEvaluation::IO::FileLogCabinet Class Reference	217
F.39.1	Detailed Description	217
F.39.2	Constructor & Destructor Documentation	217
	FileLogCabinet	217

	FileLogCabinet . . . . .	217
F.39.3	Member Function Documentation . . . . .	218
	getCount . . . . .	218
	getDescription . . . . .	218
	getPathname . . . . .	218
	newLogsheet . . . . .	218
F.40	BiometricEvaluation::IO::FileLogsheet Class Reference . . . . .	218
F.40.1	Detailed Description . . . . .	219
F.40.2	Constructor & Destructor Documentation . . . . .	220
	FileLogsheet . . . . .	220
	FileLogsheet . . . . .	220
	~FileLogsheet . . . . .	220
	FileLogsheet . . . . .	220
F.40.3	Member Function Documentation . . . . .	220
	mergeLogsheets . . . . .	220
	operator= . . . . .	221
	sequence . . . . .	221
	sync . . . . .	221
	trim . . . . .	221
	updateCursor . . . . .	222
	write . . . . .	222
	writeComment . . . . .	222
	writeDebug . . . . .	222
F.40.4	Member Data Documentation . . . . .	223
	.cursor . . . . .	223
	.sequenceFile . . . . .	223
	.theLogFile . . . . .	223
	BE_FILELOGSHEET_SEQ_NEXT . . . . .	223
	BE_FILELOGSHEET_SEQ_START . . . . .	223
F.41	BiometricEvaluation::IO::FileRecordStore Class Reference . . . . .	223
F.41.1	Detailed Description . . . . .	224
F.41.2	Constructor & Destructor Documentation . . . . .	224
	FileRecordStore . . . . .	224
	FileRecordStore . . . . .	224
F.41.3	Member Function Documentation . . . . .	225
	flush . . . . .	225
	getSpaceUsed . . . . .	225
	insert . . . . .	225
	length . . . . .	226
	move . . . . .	226
	read . . . . .	226
	remove . . . . .	227
	replace . . . . .	227
	sequence . . . . .	227
	setCursorAtKey . . . . .	228
F.42	BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem Struct Reference . . . . .	228
F.42.1	Detailed Description . . . . .	228
F.42.2	Member Data Documentation . . . . .	228
	equipment . . . . .	228
	method . . . . .	229
	name . . . . .	229



F.43	BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition Struct Reference	229
F.43.1	Detailed Description	229
F.43.2	Constructor & Destructor Documentation	229
	FingerSegmentPosition	229
F.43.3	Member Data Documentation	229
	coordinates	229
	fingerPosition	229
F.44	BiometricEvaluation::Process::ForkManager Class Reference	230
F.44.1	Detailed Description	231
F.44.2	Constructor & Destructor Documentation	231
	ForkManager	231
F.44.3	Member Function Documentation	231
	addWorker	231
	broadcastSignal	231
	defaultExitCallback	231
	getIsWorkingStatus	231
	responsibleFor	232
	setExitCallback	232
	setNotWorking	232
	startWorker	232
	startWorkers	233
	stopWorker	233
	waitForWorkerExit	234
F.44.4	Member Data Documentation	234
	FORKMANAGERS	234
F.45	BiometricEvaluation::Process::ForkWorkerController Class Reference	234
F.45.1	Detailed Description	235
F.45.2	Member Function Documentation	235
	._stop	235
	everWorked	235
	getPID	235
	isWorking	236
	reset	236
F.45.3	Friends And Related Function Documentation	236
	ForkManager::addWorker	236
	ForkManager::startWorker	236
	ForkManager::startWorkers	236
	ForkManager::stopWorker	237
F.46	BiometricEvaluation::IO::GZip Class Reference	237
F.46.1	Detailed Description	238
F.46.2	Constructor & Destructor Documentation	238
	GZip	238
F.46.3	Member Function Documentation	239
	compress	239
	compress	239
	compress	239
	compress	240
	compress	240
	compress	240
	decompress	241
	decompress	241

	decompress	241
	decompress	242
	decompress	242
	decompress	242
	operator=	243
F.46.4	Member Data Documentation	243
	CHUNK_SIZE	243
	COMPRESSION_LEVEL	243
	COMPRESSION_METHOD	243
	COMPRESSION_STRATEGY	243
	INPUT_DATA_TYPE	243
	MEMORY_LEVEL	243
	WINDOW_BITS	243
F.47	BiometricEvaluation::Image::Image Class Reference	244
F.47.1	Detailed Description	245
F.47.2	Constructor & Destructor Documentation	245
	Image	245
	Image	246
F.47.3	Member Function Documentation	246
	getCompressionAlgorithm	246
	getCompressionAlgorithm	246
	getCompressionAlgorithm	247
	getCompressionAlgorithm	248
	getData	248
	getDataPointer	248
	getDataSize	248
	getDepth	249
	getDimensions	249
	getRawData	249
	getRawGrayscaleData	249
	getResolution	250
	openImage	250
	openImage	250
	openImage	250
	setDepth	251
	setDimensions	251
	setResolution	251
	valueInColorspace	251
F.47.4	Member Data Documentation	251
	bitsPerComponent	251
F.48	BiometricEvaluation::Feature::INCITSMinutiae Class Reference	252
F.48.1	Detailed Description	253
F.48.2	Constructor & Destructor Documentation	253
	INCITSMinutiae	253
F.48.3	Member Function Documentation	254
	setCorePointSet	254
	setDeltaPointSet	254
	setMinutiaPoints	254
	setRidgeCountItems	254
F.49	BiometricEvaluation::Face::INCITSView Class Reference	254
F.49.1	Detailed Description	256

F.49.2	Constructor & Destructor Documentation	256
	INCITSView	256
	INCITSView	256
F.49.3	Member Function Documentation	256
	getColorSpace	256
	getDeviceType	257
	getEyeColor	257
	getFeaturePointSet	257
	getFIDData	257
	getGender	257
	getHairColor	257
	getImageDataType	258
	getImageType	258
	getPoseAngle	258
	getPropertySet	258
	getSourceType	258
	propertiesConsidered	258
	readFaceView	258
	readHeader	259
F.50	BiometricEvaluation::Iris::INCITSView Class Reference	259
F.50.1	Detailed Description	261
F.50.2	Constructor & Destructor Documentation	261
	INCITSView	261
	INCITSView	261
F.50.3	Member Function Documentation	262
	getCameraRange	262
	getCaptureDateString	262
	getCaptureDeviceTechnology	262
	getCaptureDeviceType	262
	getCaptureDeviceVendor	262
	getCertificationFlag	262
	getEyeLabel	263
	getIIRData	263
	getImageProperties	263
	getImageType	263
	getIrisCenterInfo	263
	getQualitySet	264
	getRollAngleInfo	264
	readHeader	264
	readIrisView	264
F.51	BiometricEvaluation::Finger::INCITSView Class Reference	266
F.51.1	Detailed Description	268
F.51.2	Constructor & Destructor Documentation	268
	INCITSView	268
	INCITSView	268
F.51.3	Member Function Documentation	269
	convertImpression	269
	convertPosition	270
	getCaptureEquipmentID	270
	getFIRData	270
	getFMRData	270

	getImpressionType . . . . .	271
	getPosition . . . . .	271
	getQuality . . . . .	271
	isAppendixFCompliant . . . . .	271
	readCoreDeltaData . . . . .	271
	readExtendedDataBlock . . . . .	271
	readFMRHeader . . . . .	272
	readFVMR . . . . .	272
	readMinutiaeDataPoints . . . . .	272
	readRidgeCountData . . . . .	273
	setAppendixFCompliance . . . . .	273
	setCaptureEquipmentID . . . . .	273
	setCBEFFProductIDs . . . . .	273
	setImpressionType . . . . .	273
	setMinutiaeData . . . . .	274
	setPosition . . . . .	274
	setQuality . . . . .	274
	setViewNumber . . . . .	274
F.52	BiometricEvaluation::Memory::IndexedBuffer Class Reference . . . . .	274
F.52.1	Detailed Description . . . . .	275
F.52.2	Constructor & Destructor Documentation . . . . .	275
	IndexedBuffer . . . . .	275
F.52.3	Member Function Documentation . . . . .	276
	getIndex . . . . .	276
	getSize . . . . .	276
	operator[] . . . . .	276
	operator[] . . . . .	276
	scan . . . . .	276
	scanBeU16Val . . . . .	277
	scanBeU32Val . . . . .	277
	scanU16Val . . . . .	277
	scanU32Val . . . . .	277
	scanU64Val . . . . .	278
	scanU8Val . . . . .	278
	setIndex . . . . .	278
F.53	BiometricEvaluation::Face::ISO2005View Class Reference . . . . .	278
F.53.1	Detailed Description . . . . .	279
F.53.2	Constructor & Destructor Documentation . . . . .	279
	ISO2005View . . . . .	279
	ISO2005View . . . . .	279
F.53.3	Member Function Documentation . . . . .	280
	readISOHeader . . . . .	280
F.54	BiometricEvaluation::Finger::ISO2005View Class Reference . . . . .	280
F.54.1	Detailed Description . . . . .	281
F.54.2	Constructor & Destructor Documentation . . . . .	281
	ISO2005View . . . . .	281
	ISO2005View . . . . .	281
F.54.3	Member Function Documentation . . . . .	281
	readCoreDeltaData . . . . .	281
F.55	BiometricEvaluation::Iris::ISO2011View Class Reference . . . . .	282
F.55.1	Detailed Description . . . . .	282

F.55.2	Constructor & Destructor Documentation	283
ISO2011View		283
ISO2011View		284
F.56	BiometricEvaluation::Image::JPEG Class Reference	284
F.56.1	Detailed Description	285
F.56.2	Member Function Documentation	285
getRawData		285
getRawGrayscaleData		285
isJPEG		285
F.57	BiometricEvaluation::Image::JPEG2000 Class Reference	286
F.57.1	Detailed Description	286
F.57.2	Constructor & Destructor Documentation	286
JPEG2000		286
F.57.3	Member Function Documentation	287
getRawData		287
getRawGrayscaleData		287
isJPEG2000		287
F.58	BiometricEvaluation::Image::JPEG2000 Class Reference	288
F.58.1	Detailed Description	288
F.58.2	Member Function Documentation	288
getRawData		288
getRawGrayscaleData		289
isJPEG2000		289
F.59	BiometricEvaluation::IO::ListRecordStore Class Reference	289
F.59.1	Detailed Description	290
F.59.2	Constructor & Destructor Documentation	290
ListRecordStore		290
~ListRecordStore		290
F.59.3	Member Function Documentation	291
flush		291
getSpaceUsed		292
insert		292
length		292
move		293
read		293
remove		293
replace		294
sequence		294
setCursorAtKey		294
sync		295
F.59.4	Member Data Documentation	295
KEYLISTFILENAME		295
SOURCERECORDSTOREPROPERTY		295
F.60	BiometricEvaluation::IO::Logsheet Class Reference	295
F.60.1	Detailed Description	297
F.60.2	Member Enumeration Documentation	297
Kind		297
F.60.3	Constructor & Destructor Documentation	297
~Logsheet		297
F.60.4	Member Function Documentation	298
getAutoSync		298

	getCommentCommit . . . . .	298
	getCommit . . . . .	298
	getCurrentEntry . . . . .	298
	getCurrentEntryNumber . . . . .	298
	getCurrentEntryNumberAsString . . . . .	298
	getDebugCommit . . . . .	299
	getTypeFromURL . . . . .	299
	lineIsComment . . . . .	300
	lineIsDebug . . . . .	300
	lineIsEntry . . . . .	300
	newEntry . . . . .	300
	resetCurrentEntry . . . . .	301
	setAutoSync . . . . .	301
	setCommentCommit . . . . .	301
	setCommit . . . . .	301
	setDebugCommit . . . . .	301
	sync . . . . .	301
	trim . . . . .	302
	write . . . . .	302
	writeComment . . . . .	302
	writeDebug . . . . .	302
F.60.5	Member Data Documentation . . . . .	303
	CommentDelimiter . . . . .	303
	DebugDelimiter . . . . .	303
	DescriptionTag . . . . .	303
	EntryDelimiter . . . . .	303
	FILEURLSCHEME . . . . .	303
	SYSLOGURLSCHEME . . . . .	303
F.61	BiometricEvaluation::Process::Manager Class Reference . . . . .	303
F.61.1	Detailed Description . . . . .	304
F.61.2	Member Function Documentation . . . . .	305
	addWorker . . . . .	305
	broadcastMessage . . . . .	306
	getNextMessage . . . . .	306
	getNumActiveWorkers . . . . .	306
	getNumCompletedWorkers . . . . .	307
	getTotalWorkers . . . . .	307
	reset . . . . .	307
	startWorker . . . . .	307
	startWorkers . . . . .	308
	stopWorker . . . . .	309
	waitForMessage . . . . .	309
	waitForWorkerExit . . . . .	309
F.61.3	Member Data Documentation . . . . .	310
	_pendingExit . . . . .	310
	_workers . . . . .	310
F.62	BiometricEvaluation::IO::ManifestEntry Struct Reference . . . . .	310
F.62.1	Detailed Description . . . . .	310
F.62.2	Member Data Documentation . . . . .	310
	offset . . . . .	310
	size . . . . .	310

F.63	BiometricEvaluation::Error::MemoryError Class Reference	310
F.63.1	Detailed Description	311
F.63.2	Constructor & Destructor Documentation	311
	MemoryError	311
	MemoryError	311
F.64	BiometricEvaluation::Process::MessageCenter Class Reference	311
F.64.1	Detailed Description	311
F.64.2	Constructor & Destructor Documentation	312
	MessageCenter	312
F.64.3	Member Function Documentation	313
	disconnectClient	313
	getNextMessage	313
	hasUnseenMessages	313
	sendResponse	313
F.64.4	Member Data Documentation	313
	CONNECTION_BACKLOG	313
	DEFAULT_PORT	314
	DEFAULT_TIMEOUT	314
	MAX_MESSAGE_LENGTH	314
F.65	BiometricEvaluation::Process::MessageCenterListener Class Reference	314
F.65.1	Detailed Description	314
F.65.2	Member Function Documentation	314
	workerMain	314
F.65.3	Member Data Documentation	315
	PARAM_PORT	315
F.66	BiometricEvaluation::Process::MessageCenterReceiver Class Reference	315
F.66.1	Detailed Description	315
F.66.2	Constructor & Destructor Documentation	315
	MessageCenterReceiver	315
	~MessageCenterReceiver	316
F.66.3	Member Function Documentation	316
	workerMain	316
F.66.4	Member Data Documentation	316
	MSG_DISCONNECT	316
	PARAM_CLIENT_ID	316
	PARAM_CLIENT_SOCKET	316
F.67	BiometricEvaluation::MPI::MessageTag Class Reference	316
F.67.1	Detailed Description	316
F.67.2	Member Enumeration Documentation	316
	Kind	316
F.68	BiometricEvaluation::Feature::Minutiae Class Reference	317
F.68.1	Detailed Description	317
F.69	BiometricEvaluation::Feature::MinutiaPoint Struct Reference	317
F.69.1	Detailed Description	318
F.70	BiometricEvaluation::Feature::MPEGFacePoint Struct Reference	318
F.70.1	Detailed Description	318
F.71	BiometricEvaluation::Image::NetPBM Class Reference	318
F.71.1	Detailed Description	319
F.71.2	Member Function Documentation	319
	ASCIIBitmapTo8Bit	319
	ASCIIPixmapToBinaryPixmap	320

BinaryBitmapTo8Bit	320
getNextValue	320
getRawData	321
getRawGrayscaleData	321
isNetPBM	322
skipComment	323
skipLine	323
F.72 BiometricEvaluation::Error::NotImplemented Class Reference	323
F.72.1 Detailed Description	324
F.72.2 Constructor & Destructor Documentation	324
NotImplemented	324
NotImplemented	324
F.73 BiometricEvaluation::Error::ObjectDoesNotExist Class Reference	324
F.73.1 Detailed Description	324
F.73.2 Constructor & Destructor Documentation	324
ObjectDoesNotExist	324
ObjectDoesNotExist	324
F.74 BiometricEvaluation::Error::ObjectExists Class Reference	325
F.74.1 Detailed Description	325
F.74.2 Constructor & Destructor Documentation	325
ObjectExists	325
ObjectExists	325
F.75 BiometricEvaluation::Error::ObjectIsClosed Class Reference	325
F.75.1 Detailed Description	326
F.75.2 Constructor & Destructor Documentation	326
ObjectIsClosed	326
ObjectIsClosed	326
F.76 BiometricEvaluation::Error::ObjectIsOpen Class Reference	326
F.76.1 Detailed Description	326
F.76.2 Constructor & Destructor Documentation	326
ObjectIsOpen	326
ObjectIsOpen	326
F.77 BiometricEvaluation::Memory::OrderedMap< Key, T > Class Template Reference	327
F.77.1 Detailed Description	328
F.77.2 Constructor & Destructor Documentation	328
OrderedMap	328
~OrderedMap	328
F.77.3 Member Function Documentation	328
begin	328
begin	328
cbegin	328
cend	328
end	328
end	329
erase	329
erase	329
find	329
key_eq	329
keyExists	329
operator[]	330
push_back	330



size	330
F.78 BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > Class Template Reference	330
F.78.1 Detailed Description	331
F.78.2 Constructor & Destructor Documentation	331
OrderedMapConstIterator	331
OrderedMapConstIterator	332
~OrderedMapConstIterator	332
F.78.3 Member Function Documentation	332
operator"!="	332
operator*	332
operator++	332
operator++	332
operator--	332
operator--	332
operator->	333
operator==	333
F.79 BiometricEvaluation::Memory::OrderedMapIterator< Key, T > Class Template Reference	333
F.79.1 Detailed Description	334
F.79.2 Constructor & Destructor Documentation	334
OrderedMapIterator	334
~OrderedMapIterator	334
F.79.3 Member Function Documentation	334
operator"!="	334
operator*	334
operator++	334
operator++	335
operator--	335
operator--	335
operator->	335
operator==	335
F.80 BiometricEvaluation::Error::ParameterError Class Reference	335
F.80.1 Detailed Description	336
F.80.2 Constructor & Destructor Documentation	336
ParameterError	336
ParameterError	336
F.81 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification Class Reference	336
F.81.1 Detailed Description	336
F.82 BiometricEvaluation::Image::PNG Class Reference	336
F.82.1 Detailed Description	337
F.82.2 Member Function Documentation	337
getRawData	337
getRawGrayscaleData	337
isPNG	337
F.83 BiometricEvaluation::Face::PoseAngle Struct Reference	338
F.83.1 Detailed Description	338
F.84 BiometricEvaluation::Process::POSIXThreadManager Class Reference	338
F.84.1 Detailed Description	339
F.84.2 Constructor & Destructor Documentation	339
POSIXThreadManager	339
F.84.3 Member Function Documentation	339
addWorker	339

startWorker	339
startWorkers	339
stopWorker	340
waitForWorkerExit	340
F.85 BiometricEvaluation::Process::POSIXThreadWorkerController Class Reference	340
F.85.1 Detailed Description	341
F.85.2 Member Function Documentation	341
everWorked	341
isWorking	341
reset	341
F.86 BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate Struct Reference	342
F.86.1 Detailed Description	342
F.86.2 Constructor & Destructor Documentation	342
PrintPositionCoordinate	342
F.86.3 Member Data Documentation	342
coordinates	342
fingerView	342
segment	343
F.87 BiometricEvaluation::IO::Properties Class Reference	343
F.87.1 Detailed Description	344
F.87.2 Constructor & Destructor Documentation	344
Properties	344
Properties	344
~Properties	344
F.87.3 Member Function Documentation	344
getMode	344
getProperty	344
getPropertyAsDouble	345
getPropertyAsInteger	345
getPropertyKeys	345
initWithBuffer	345
initWithBuffer	346
removeProperty	346
setProperty	346
setPropertyFromDouble	346
setPropertyFromInteger	348
F.88 BiometricEvaluation::IO::PropertiesFile Class Reference	348
F.88.1 Detailed Description	349
F.88.2 Constructor & Destructor Documentation	349
PropertiesFile	349
~PropertiesFile	349
PropertiesFile	349
F.88.3 Member Function Documentation	349
changeName	349
operator=	350
sync	350
F.89 BiometricEvaluation::Iris::INCITSView::QualitySubBlock Struct Reference	350
F.89.1 Detailed Description	350
F.90 BiometricEvaluation::Image::Raw Class Reference	351
F.90.1 Detailed Description	351

F.90.2	Member Function Documentation	351
getRawData		351
getRawGrayscaleData		351
F.91	BiometricEvaluation::MPI::Receiver Class Reference	352
F.91.1	Detailed Description	352
F.91.2	Constructor & Destructor Documentation	353
Receiver		353
F.91.3	Member Function Documentation	354
start		354
F.92	BiometricEvaluation::MPI::RecordProcessor Class Reference	354
F.92.1	Detailed Description	355
F.92.2	Constructor & Destructor Documentation	355
RecordProcessor		355
F.92.3	Member Function Documentation	355
newProcessor		355
performInitialization		356
processRecord		356
processRecord		356
processWorkPackage		356
F.93	BiometricEvaluation::IO::RecordStore Class Reference	357
F.93.1	Detailed Description	359
F.93.2	Member Enumeration Documentation	359
Kind		359
F.93.3	Constructor & Destructor Documentation	359
RecordStore		359
RecordStore		360
F.93.4	Member Function Documentation	360
begin		360
changeDescription		360
containsKey		360
createRecordStore		361
end		362
flush		362
genKeySegName		362
getCount		363
getDescription		363
getPathname		363
getProperties		363
getSpaceUsed		363
insert		363
insert		364
length		364
mergeRecordStores		364
move		365
openRecordStore		365
read		366
read		366
remove		366
removeRecordStore		367
replace		367
replace		367

	sequence	368
	sequence	368
	setCursorAtKey	369
	setProperty	369
	sync	369
F.93.5	Member Data Documentation	370
	BE_RECSTORE_SEQ_NEXT	370
	BE_RECSTORE_SEQ_START	370
	CONTROLFILENAME	370
	COUNTPROPERTY	370
	DESCRIPTIONPROPERTY	370
	INVALIDKEYCHARS	370
	KEY_SEGMENT_SEPARATOR	370
	KEY_SEGMENT_START	370
	RSREADONLYERROR	370
	TYPEPROPERTY	370
F.94	BiometricEvaluation::MPI::RecordStoreDistributor Class Reference	371
F.94.1	Detailed Description	371
F.94.2	Constructor & Destructor Documentation	371
	RecordStoreDistributor	371
F.94.3	Member Function Documentation	372
	createWorkPackage	372
F.95	BiometricEvaluation::IO::RecordStoreIterator Class Reference	372
F.95.1	Detailed Description	373
F.95.2	Constructor & Destructor Documentation	373
	RecordStoreIterator	373
	RecordStoreIterator	373
	RecordStoreIterator	373
	RecordStoreIterator	373
	~RecordStoreIterator	374
F.95.3	Member Function Documentation	374
	operator"!="	374
	operator*	374
	operator+	374
	operator++	374
	operator++	374
	operator+=	374
	operator->	375
	operator=	375
	operator==	375
F.96	BiometricEvaluation::MPI::RecordStoreResources Class Reference	375
F.96.1	Detailed Description	376
F.96.2	Constructor & Destructor Documentation	376
	RecordStoreResources	376
F.96.3	Member Function Documentation	376
	getOptionalProperties	376
	getRecordStore	376
	getRequiredProperties	377
	haveRecordStore	377
F.97	BiometricEvaluation::Image::Resolution Struct Reference	377
F.97.1	Detailed Description	377

F.97.2	Member Enumeration Documentation	377
	Units	377
F.97.3	Constructor & Destructor Documentation	378
	Resolution	378
F.97.4	Member Data Documentation	379
	units	379
	xRes	379
	yRes	379
F.98	BiometricEvaluation::MPI::Resources Class Reference	379
F.98.1	Detailed Description	380
F.98.2	Constructor & Destructor Documentation	380
	Resources	380
F.98.3	Member Function Documentation	380
	getLogsheetURL	380
	getOptionalProperties	380
	getPropertiesFileName	380
	getRequiredProperties	381
F.99	BiometricEvaluation::Feature::RidgeCountItem Struct Reference	381
F.99.1	Detailed Description	381
F.100	BiometricEvaluation::MPI::Runtime Class Reference	381
F.100.1	Detailed Description	382
F.100.2	Constructor & Destructor Documentation	382
	Runtime	382
F.100.3	Member Function Documentation	382
	abort	382
	shutdown	382
	start	382
F.101	BiometricEvaluation::Process::Semaphore Class Reference	382
F.101.1	Detailed Description	383
F.101.2	Constructor & Destructor Documentation	383
	Semaphore	383
	Semaphore	383
F.101.3	Member Function Documentation	384
	post	384
	timedwait	385
	trywait	385
	wait	385
F.102	BiometricEvaluation::Error::SignalManager Class Reference	386
F.102.1	Detailed Description	386
F.102.2	Constructor & Destructor Documentation	387
	SignalManager	387
	SignalManager	387
F.102.3	Member Function Documentation	387
	clearSigHandled	387
	clearSignalSet	387
	setDefaultSignalSet	387
	setSigHandled	387
	setSignalSet	387
	sigHandled	388
	start	388
	stop	388

F.102.4 Member Data Documentation	388
.canSigJump	388
.sigJumpBuf	388
F.103 BiometricEvaluation::Image::Size Struct Reference	389
F.103.1 Detailed Description	389
F.103.2 Constructor & Destructor Documentation	389
Size	389
F.103.3 Member Data Documentation	389
xSize	389
ySize	389
F.104 BiometricEvaluation::IO::SQLiteRecordStore Class Reference	389
F.104.1 Detailed Description	390
F.104.2 Member Function Documentation	390
changeDescription	390
cleanup	391
createKeyValueTable	391
createStructure	391
flush	391
getSpaceUsed	392
insert	392
length	392
move	392
read	393
readSegments	393
remove	393
replace	394
sequence	394
setCursorAtKey	395
sqliteError	396
validateKeyValueTable	396
validateSchema	396
F.105 BiometricEvaluation::Process::Statistics Class Reference	396
F.105.1 Detailed Description	397
F.105.2 Constructor & Destructor Documentation	397
Statistics	397
Statistics	397
F.105.3 Member Function Documentation	398
callStatistics_logStats	398
getCPUTimes	398
getMemorySizes	398
getNumThreads	398
logStats	399
startAutoLogging	399
stopAutoLogging	399
F.106 BiometricEvaluation::Error::StrategyError Class Reference	400
F.106.1 Detailed Description	400
F.106.2 Constructor & Destructor Documentation	400
StrategyError	400
StrategyError	400
F.107 BiometricEvaluation::IO::SysLogsheet Class Reference	400
F.107.1 Detailed Description	401

F.107.2 Constructor & Destructor Documentation	402
SysLogsheet	402
SysLogsheet	403
~SysLogsheet	403
SysLogsheet	403
F.107.3 Member Function Documentation	403
operator=	403
setup	404
sync	404
write	404
writeComment	404
writeDebug	404
writeToLogger	405
F.107.4 Member Data Documentation	405
_operational	405
_sequenced	405
_sockFD	405
_utc	405
F.108 BiometricEvaluation::MPI::TaskCommand Class Reference	405
F.108.1 Detailed Description	405
F.108.2 Member Enumeration Documentation	405
Kind	405
F.109 BiometricEvaluation::MPI::TaskStatus Class Reference	406
F.109.1 Detailed Description	406
F.109.2 Member Enumeration Documentation	406
Kind	406
F.110 BiometricEvaluation::Time::Timer Class Reference	406
F.110.1 Detailed Description	406
F.110.2 Member Typedef Documentation	407
BE_CLOCK_TYPE	407
F.110.3 Constructor & Destructor Documentation	407
Timer	407
F.110.4 Member Function Documentation	407
elapsed	407
start	407
stop	407
F.111 BiometricEvaluation::View::View Class Reference	407
F.111.1 Detailed Description	408
F.111.2 Member Function Documentation	408
getCompressionAlgorithm	408
getImage	409
getImageDepth	409
getImageResolution	409
getImageSize	409
getScanResolution	409
setImageData	410
setImageDepth	411
setImageResolution	411
setImageSize	411
setScanResolution	411
F.112 BiometricEvaluation::Time::Watchdog Class Reference	411

F.112.1 Detailed Description . . . . .	412
F.112.2 Constructor & Destructor Documentation . . . . .	413
Watchdog . . . . .	413
F.112.3 Member Function Documentation . . . . .	414
clearCanSigJump . . . . .	414
clearExpired . . . . .	414
expired . . . . .	414
setCanSigJump . . . . .	414
setExpired . . . . .	414
setInterval . . . . .	414
start . . . . .	414
stop . . . . .	415
F.112.4 Member Data Documentation . . . . .	415
PROCESSTIME . . . . .	415
REALTIME . . . . .	415
F.113 BiometricEvaluation::Process::Worker Class Reference . . . . .	415
F.113.1 Detailed Description . . . . .	416
F.113.2 Member Function Documentation . . . . .	416
_initCommunication . . . . .	416
closeManagerPipeEnds . . . . .	416
closeWorkerPipeEnds . . . . .	417
getParameter . . . . .	417
getParameterAsDouble . . . . .	417
getParameterAsInteger . . . . .	417
getParameterAsString . . . . .	418
getReceivingPipe . . . . .	418
getSendingPipe . . . . .	418
receiveMessageFromManager . . . . .	419
sendMessageToManager . . . . .	419
setParameter . . . . .	419
stop . . . . .	419
stopRequested . . . . .	420
waitForMessage . . . . .	420
workerMain . . . . .	420
F.114 BiometricEvaluation::Process::WorkerController Class Reference . . . . .	420
F.114.1 Detailed Description . . . . .	421
F.114.2 Constructor & Destructor Documentation . . . . .	421
WorkerController . . . . .	421
F.114.3 Member Function Documentation . . . . .	422
everWorked . . . . .	422
finishedWorking . . . . .	422
getWorker . . . . .	422
isWorking . . . . .	422
reset . . . . .	422
sendMessageToWorker . . . . .	423
setParameter . . . . .	423
setParameterFromDouble . . . . .	423
setParameterFromInteger . . . . .	423
setParameterFromString . . . . .	424
F.114.4 Member Data Documentation . . . . .	424
_worker . . . . .	424



F.115BiometricEvaluation::MPI::WorkPackage Class Reference . . . . .	424
F.115.1 Detailed Description . . . . .	425
F.115.2 Constructor & Destructor Documentation . . . . .	425
WorkPackage . . . . .	425
F.115.3 Member Function Documentation . . . . .	425
getNumElements . . . . .	425
getSize . . . . .	425
setData . . . . .	425
setNumElements . . . . .	425
F.116BiometricEvaluation::MPI::WorkPackageProcessor Class Reference . . . . .	425
F.116.1 Detailed Description . . . . .	426
F.116.2 Member Function Documentation . . . . .	426
getLogsheet . . . . .	426
newProcessor . . . . .	426
performInitialization . . . . .	427
processWorkPackage . . . . .	427
setLogsheet . . . . .	427
F.117BiometricEvaluation::Image::WSQ Class Reference . . . . .	427
F.117.1 Detailed Description . . . . .	428
F.117.2 Member Function Documentation . . . . .	428
getRawData . . . . .	428
getRawGrayscaleData . . . . .	428
isWSQ . . . . .	429
<b>Index</b>	<b>430</b>

# 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 [14].

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. 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 program 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 each package and includes example code. The long form of this document includes 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++[16] 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;
- Reduce memory allocation errors;
- Simplify the use of parallel processing.

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 [3] 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.

Applications can use the `Messaging` package to communicate between threads and processes, or to a terminal. Messages in this context are simply an array of bytes. One such use could be providing a command line interface to an long-running process.

The `MPI` package provides wrappers around the Message Passing Interface (MPI) [12] libraries, handling all MPI communication and error events. Many parallel applications can be greatly simplified, only implementing a few methods to process data.

BECommon is designed to be used in a modular fashion, and it is possible to compile many 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, as well as to provide services through general purpose utility functions to other parts of the framework.

### 3.1 Versioning

Version numbers, the compiler used, and other framework metadata 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  /* "Framework Version: 0.4" */
2  std::cout << "Framework Version: " << BE::Framework::getMajorVersion() << "." <<
3      BE::Framework::getMinorVersion() << std::endl;
4
5  /* "Compiler Used: clang v5.1.0" */
6  std::cout << "Compiler Used: " << BE::Framework::getCompiler() << " v" <<
7      BE::Framework::getCompilerVersion() << std::endl;
8
9  /* "Date/Time Compiled: Jan 24 2014 12:16:01" */
10 std::cout << "Date/Time Compiled: " << BE::Framework::getCompileDate() << " " <<
11     BE::Framework::getCompileTime() << std::endl;
```

### 3.2 Enumerations

As of C++ 2011, `enum s` can be strongly-typed. The Biometric Evaluation Framework makes use of these strongly-typed `enum class` es throughout. As an added convenience, functions converting to and from `enum s`, `string s`, and `int s` are implicitly implemented easily via a template, eliminating many lines of boiler-plate code and creating equivalence in functionality among `enum class` es throughout `BECommon`.

At the core of `Framework::Enumeration` is a `const` mapping of `enum` to `string`, defined by you in code and instantiated at compile-time. As demonstrated in Listing 3.2, simply define your `enum class` and populate the map.

Listing 3.2: `Framework::Enumeration`

```
1  /*
2  * color.h
```

```
3  */
4
5  enum class Color
6  {
7      Black,
8      Blue,
9      Green
10 };
11
12 /*
13  * color.cpp
14  */
15
16 #include <be_framework_enumeration.h>
17
18 template<>
19 const std::map<Color, std::string>
20 BiometricEvaluation::Framework::EnumerationFunctions<Color>::enumToStringMap {
21     {Color::Black, "Black"},
22     {Color::Blue, "Blue"},
23     {Color::Green, "Green"}
24 };
25
26 /*
27  * application.cpp
28  */
29
30 #include <color.h>
31
32 /* "Black" */
33 std::cout << to_string(Color::Black) << std::endl;
34 /* "2" */
35 std::cout << to_int_type(Color::Green) << std::endl;
36 /* Color::Blue */
37 Color color = to_enum<Color>("Blue");
```

While `Framework::Enumeration` was created for `BECommon`, the `template`'s only dependency is `Exception`, and so it can easily be used in other C++ 2011 projects.

## 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 [13]. 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 `AutoArray`s with `RecordStore`s

```

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" [16, 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 16](#).

### 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(asigmgr, 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.

`RecordStore`s 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 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 <be_io_dbrecstore.h>
2 int
3 main(int argc, char* argv[]) {
4
5     IO::DBRecordStore *rs;
6     try {
7         rs = new IO::DBRecordStore("myRecords", "My Record Store");
8     } catch (Error::Exception& e) {
9         cout << "Caught " << e.what() << endl;
10        return (EXIT_FAILURE);
11    }
12    std::unique_ptr<IO::DBRecordStore> urs(rs);
13
14    try {
15        Memory::uint8Array theData;
16
17        theData = getSomeData();
18        urs->insert("key1", theData);
19
20        theData = getSomeData();
21        urs->insert("key2", theData);
22
23    } catch (Error::Exception& e) {
24        cout << "Caught " << e.what() << endl;
25        return (EXIT_FAILURE);
26    }
27
28    // Some more processing where new data for a key comes in ...
29    theData = getSomeData();
30    urs->replace("key1", theData);
31
32    // Obtain the data for all keys ...
33    string theKey;
34    while (true) {
35        uint64_t len = rs->sequence(theKey, theData);
36        cout << "Read data for key " << theKey << " of length " << len << endl;
37    }
38    // The data for the key is no longer needed ...
39    urs->remove("key1");
40 }
```

## 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 storage details. Management of the log messages to the backing store is done within the `Logsheet` implementations. `Logsheet` specifies the common interface to

all implementations. In addition, objects of this class can be created to provide a “Null” Logsheet where messages are not saved.

A Logsheet is an output stream (subclass of `std::ostream`), and therefore can handle built-in types and any class that supports streaming. Each entry is numbered by the Logsheet class when written to the log. A call to the `newEntry()` method commits the current entry to the log, 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. Logsheet also supports the writing of “debug” and comment entries. Each entry is prefixed with a letter code indicating the type.

### 6.3.1 FileLogsheet

`IO::FileLogsheet` uses a file to store the log messages. Access to this file is not controlled, and therefore, if two instances of this class are made with the same file name, the results are undefined. The description of the sheet is placed at the top of the file during construction of the object. Objects of this class can be constructed with a string containing a `file://` Uniform Resource Locator (URL) or a simple file name.

`IO::FileLogCabinet` is a container of `FileLogsheet` where each log file is contained within the same directory owned by this container class.

The example code in Listing 6.2 shows how an application can use a `FileLogsheet`, contained within a `FileLogCabinet`, to record operational information.

Listing 6.2: Using a `FileLogsheet` within a `FileLogCabinet`

```

1 #include <be_io_filelogcabinet.h>
2 using namespace BiometricEvaluation;
3 using namespace BiometricEvaluation::IO;
4
5 FileLogCabinet *lc;
6 try {
7     lc = new FileLogCabinet(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.what() << endl;
13    return (-1);
14 }
15 std::unique_ptr<FileLogCabinet> ulc(lc);
16 try {
17     ls = alc->newLogsheet("log01", "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.what() << 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 unique_ptr

```

### 6.3.2 SysLogsheet

The `SysLogsheet` is an implementation of `Logsheet` which writes log entries to a system logger service. Objects of this class are created with a URL starting with `syslog://`. When using a system logger, the URL must give the hostname of the logger as well as the network port: `syslog://node00:4315` for example. The system logger must understand the Syslog protocol as specified in RFC5424 [17].

Multiple instances of a `SysLogsheet` can be created with the same URL with the assumption that the logging server can manage multiple incoming message streams.

## 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.what() << endl;
7     return;
8 } catch (Error::FileError& e) {
9     cerr << "A file error occurred: " << e.what() << 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 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     compressor->compress(largeBuffer, compressedBuffer);
8 } catch (Error::Exception &e) {
9     cerr << "Could not compress (" << e.what() << ')' << endl;
10 }
```

Different `Compressor`s 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 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 and Control

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

### 8.1 Process Statistics

When an application is running, there may be a need to obtain information of the process executing that application. The `Process` 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 16) 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 log. 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 Ustime 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. Applications can take advantage of a multi-core machine, for example. BECommon aims to simplify this by abstracting the usage of process and thread creation to run multiple instances of the same function in parallel.

### 8.2.1 Manager

There are three class hierarchies involved in the abstraction. The `BiometricEvaluation::Process::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 a worker 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 `Worker`s exit `workerMain()` 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 on the next page.

### 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 `Worker`

Controller 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 `Manager`s and `WorkerController`s.

Listing 8.4: Using `Manager`s and `WorkerController`s

```

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 }

```

### 8.2.4 Communications

Managers and workers may have a good reason to send and receive messages directly. A communications mechanism is built-in to the [Process Management](#) model to facilitate such communications. The type and content of the message is completely up to the client implementation, since messages are sent as `AutoArray s`. A manager does not directly send messages to a worker. This service is provided by the `WorkerController` (via `sendMessageToWorker()`).

Managers can keep an eye on incoming messages by calling the (optionally blocking) `waitForMessage()` method. This method will return a handle to the worker that sent a message. Alternatively, the manager can invoke `getNextMessage()` (again, blocking optional) to immediately receive the next message.

Listing 8.5 and Listing 8.6 are continuations of Listing 8.3 on page 25 and Listing 8.4 on the previous page respectively, showing an example of communication, using `std::string` messages.

Listing 8.5: Worker Communication

```

1     Memory::uint8Array msg;
2
3     /* Deal with next customer unless Manager interrupts in next second */
4     if (this->waitForMessage(1)) {
5         if (this->receiveMessageFromManager(msg)) {
6             Action action = Restaurant::messageToAction(msg);
7             switch (action) {
8                 case TAKE_BREAK:
9                     employee->goOnBreak();
10                    break;
11                    /* ... */
12                }
13            }
14        }
15
16        /* ... */
17
18        if (customer->isComplaining()) {
19            sprintf((char *)&(*msg), "Customer Complaint");
20            this->sendMessageToManager(msg);
21        }

```

Listing 8.6: Manager Communication

```

1     tr1::shared_ptr<Process::WorkerController> sender;
2     Memory::uint8Array msg;
3
4     /* Do routine tasks unless employee has concern in the next 2 seconds */
5     while (this->getNextMessage(sender, msg, 2)) {
6         Action action = Restaurant::messageToAction(msg);
7         switch (action) {

```

```
8         case CUSTOMER_COMPLAINT:
9             sprintf((char *)&(*msg), "I'll take care of it.");
10            this->sendMessageToWorker(msg);
11            break;
12        /* ... */
13    }
14
15
16    /* ... */
17
18    /* Closing Time */
19    sprintf((char *)&(*msg), "Clock out and go home.");
20    this->broadcastMessage(msg);
```



## 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 45 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  $\leq 127$ . `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` [13].

## 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`) [11]. `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` [15].

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 [18] are represented by the `WSQ` class. The WSQ decompressor found in NIST Biometric Image Software [13], `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 `BECommon` is more than warranted. A complete listing of functions is available in the documentation appendix for `BiometricEvaluation::Text2`.

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 16), 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 34).

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 45) or `DataInterchange` (Chapter 15 on page 47) 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.

Currently, fingerprint and palm print minutiae are the features supported within the `BECCommon`. As development continues, additional features contained within biometric data records will be supported.

### 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 41).

See Listing 13.1 on page 42 for a complete example of how to obtain the fingerprint minutiae data from an ANSI/NIST record.

### 12.2 ISO/INCITS Features

The ISO [2] and INCITS [1] fingerprint minutiae standards are represented within `BECCommon` with the same class, `INCITSMinutiae`, as the minutiae format is identical in both standards.

Listing 13.2 on page 43 shows how to create a view object for the fingerprint minutiae record contained in a file.





# 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` (Chapter 15 on page 47) 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.)

### 13.1 ANSI/NIST Minutiae Data Record

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 the ANSI/NIST record 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 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.

#### 13.1.1 ANSI/NIST Finger Views

An ANSI/NIST record may contain one or more finger views, each based on a type of finger image. These Type-3, Type-4, etc. records contain the image and Type-9 minutiae data, among other information. These

record types are grouped into either the fixed- or variable-resolution categories, and are represented as specific classes within BECommon, AN2KViewFixedResolution and AN2KViewVariableResolution.

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.

Listing 13.1 shows how an application can use the AN2KViewFixedResolution to retrieve image information, image data, and derived minutiae information from a file containing an ANSI/NIST record with Type-3 (fixed resolution image) and Type-9 (fingerprint minutiae) records.

**Listing 13.1: 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 }
```

```

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 | }

```

### 13.1.2 ISO/INCITS Finger Views

The ISO [10] and INCITS [9] standards typically use separate files for the source biometric data and the derived data. For example, the ISO 19794-2 standard is for fingerprint minutiae data, while 19794-4 is for finger image data. The corresponding BECommon view objects are constructed with both files, although a view can be constructed with only one file. In the latter case, the view object will represent only that information contained in the single file.

Listing 13.1 on the preceding page shows how an application can create a view from a ANSI/INCTIS 378 finger minutiae format record [1].

Listing 13.2: Using an INCITS Finger View

```

1 | #include <stdlib.h>
2 | #include <fstream>
3 | #include <iostream>
4 | #include <be_finger_ansi2004view.h>
5 | #include <be_feature_incitsminutiae.h>
6 | using namespace std;
7 | using namespace BiometricEvaluation;
8 |
9 | int
10 | main(int argc, char* argv[]) {
11 |
12 |     Finger::ANSI2004View fngv;
13 |     try {
14 |         fngv = Finger::ANSI2004View("test_data/fmr.ansi2004", "", 3);
15 |     } catch (Error::DataError &e) {
16 |         cerr << "Caught " << e.getInfo() << endl;
17 |         return (EXIT_FAILURE);
18 |     } catch (Error::FileError& e) {
19 |         cerr << "A file error occurred: " << e.getInfo() << endl;
20 |         return (EXIT_FAILURE);
21 |     }
22 |     cout << "Image resolution is " << fngv.getImageResolution() << endl;
23 |     cout << "Image size is " << fngv.getImageSize() << endl;
24 |     cout << "Image depth is " << fngv.getImageDepth() << endl;
25 |     cout << "Compression is " << fngv.getCompressionAlgorithm() << endl;
26 |     cout << "Scan resolution is " << fngv.getScanResolution() << endl;
27 |
28 |     Feature::INCITSMinutiae fmd = fngv.getMinutiaeData();
29 |     cout << "Minutiae format is " << fmd.getFormat() << endl;
30 |     Feature::MinutiaPointSet mps = fmd.getMinutiaPoints();
31 |     cout << "There are " << mps.size() << " minutiae points:" << endl;
32 |     for (size_t i = 0; i < mps.size(); i++)
33 |         cout << mps[i];
34 |
35 |     Feature::RidgeCountItemSet rcs = fmd.getRidgeCountItems();

```

```
36 |     cout << "There are " << rcs.size() << " ridge count items:" << endl;
37 |     for (int i = 0; i < rcs.size(); i++)
38 |         cout << "\t" << rcs[i];
39 |
40 |     Feature::CorePointSet cores = fmd.getCores();
41 |     cout << "There are " << cores.size() << " cores:" << endl;
42 |     for (int i = 0; i < cores.size(); i++)
43 |         cout << "\t" << cores[i];
44 |
45 |     Feature::DeltaPointSet deltas = fmd.getDeltas();
46 |     cout << "There are " << deltas.size() << " deltas:" << endl;
47 |     for (int i = 0; i < deltas.size(); i++)
48 |         cout << "\t" << deltas[i];
49 |
50 |     exit (EXIT_SUCCESS);
51 | }
```

# 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.

In the case where a raw image is part of the biometric record, the `View` object's related `Image` ([Chapter 10 on page 33](#)) 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.

`View`s are high-level abstractions of the biometric sample, and concrete implementations of a `View` include finger, face, iris, etc. views based on a specific type of biometric. Therefore, `View` objects are not created directly. Subclasses, such as finger views (see [Chapter 13 on page 41](#)), represent the specific type of biometric sample.

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.



## Chapter 15

# Data Interchange

The `DataInterchange` package consists of classes and other elements used to process an entire biometric data record, or set of records. 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 `Finger Views` Chapter 13 on page 41) from this 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 Finger Captures

```
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 Latent 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                 trl::shared_ptr<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) 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 13.2 on page 43 shows an example of accessing the information in an ANSI 378-2004 Finger Minutiae Record by creating an ANSI2004View object from the record file.

# Chapter 16

## Messaging

Biometric Evaluation Framework contains a collection of classes to facilitate receiving messages asynchronously over a network. What is done with these messages and how (or if) to respond is ultimately up to the application. BECommon uses this messaging in a concrete way to receive text-based commands from a `telnet` session over the Internet.

### 16.1 Message Center

`Process::MessageCenter` is the public-facing class an application uses to receive messages over a network. A *message* is a user-defined blob of data stored in an array of bytes. Instantiate a `MessageCenter`, and it will diligently await connections on the specified port in a separate process. During its run-loop, the application may poll or wait to determine if a message is waiting. The application has the choice of dealing with the message, sending a response, or ignoring the message entirely. Because the `MessageCenterListener` is in a separate process, the main run-loop of the application does not have to be interrupted. The `MessageCenter` classes utilize existing framework inter-process communication techniques to propagate messages (see Subsection 8.2.4 on page 28).

Listing 16.1: Basic MessageCenter Usage

```
1 namespace BE = BiometricEvaluation;
2
3 uint32_t clientID;
4 BE::Memory::uint8Array message;
5 BE::Process::MessageCenter mc;
6 for (;;) {
7     /* ... do work ... */
8
9     if (mc.hasUnseenMessages()) {
10         mc.getNextMessage(clientID, message);
11         std::cout << clientID << " sent a " << message.size() <<
12             " byte message." << std::endl;
13
14         Memory::AutoArrayUtility::setString(message, "ACK\n");
15         mc.sendResponse(clientID, message);
16     }
17 }
```

Messages can be sent to the `MessageCenter` in a number of ways, like `telnet` connections or `write()` ing to a socket. Messages are terminated with a newline (`\n`) character.

## 16.2 Command Center

It's easy to see how `MessageCenter` might be used for passing *commands* to a running application. One might want to query the *status* of an operation or ask a process to *stop*. The aim of `CommandCenter` was to take this common command-passing pattern and make it easier.

With `CommandCenter`, an application defines one or more `enum class` es using `Framework::Enumerations` (see Section 3.2 on page 5). For convenience, the application should subclass the `CommandParser` template, with the enumeration as the templated type. The base class instantiates a `MessageCenter` and listens for connections. Just like `MessageCenter`, commands do not have to be dealt with or responded to, and the application will only know if a command is awaiting a response if the application asks.

Because `CommandParser` operates off of strongly-typed enumerations, a pure virtual method, `parse(Command)`, must be implemented in the child class. It is expected that this method will simply be a `switch` statement of all possible enumerations (*commands*). The body of the `switch` will likely call other methods, each dealing with a single command.

`CommandParser` performs some additional convenience functions to help application developers quickly respond to commands. A *usage* string may be automatically sent when an invalid command is received. The application's main run-loop will never see the failed command attempt. If a valid command is received, `CommandParser` will tokenize any extra text in the sent command and store it in an easily retrieved `vector`. The method called from `parse()` can then sanity-check the arguments and send an error message back to the client if the arguments are invalid.

Listing 16.2: Basic `CommandCenter` Usage

```

1 namespace BE = BiometricEvaluation;
2
3 enum class TestCommand
4 {
5     Stop,
6     Help
7 };
8
9 template<>
10 const std::map<TestCommand, std::string>
11 BE::Framework::EnumerationFunctions<TestCommand>::enumToStringMap {
12     {TestCommand::Stop, "STOP"},
13     {TestCommand::Help, "HELP"}
14 };
15
16 class TestCommandParser : public BE::Process::CommandParser<TestCommand>
17 {
18 public:
19     void
20     parse(
21         const BE::Process::CommandParser<TestCommand>::Command &command)
22     {
23         switch (command.command) {
24             case TestCommand::Stop:
25                 this->stop(command);
26                 break;
27             case TestCommand::Help:
28                 this->help(command);
29                 break;
30         }
31     }
32

```

```

33 private:
34     void
35     stop(
36         const BE::Process::CommandParser<TestCommand>::Command &command)
37     {
38         /* Ensure proper arguments */
39         if (command.arguments.size() != 1) {
40             this->sendResponse(command.clientID, "Usage: " +
41                 to_string(command.command) + " <process>");
42             return;
43         }
44
45         /* ... perform stop operation ... */
46     }
47
48     void
49     help(
50         const BE::Process::CommandParser<TestCommand>::Command &command)
51     {
52         this->sendResponse(command.clientID, "Available Commands:\n"
53             "\tSTOP <process>\n\tHELP");
54     }
55 };
56
57 int
58 main()
59 {
60     TestCommandParser commandCenter;
61     TestCommandParser::Command command;
62     for (;;) {
63         /* ... do work ... */
64
65         if (commandCenter.hasPendingCommands()) {
66             commandCenter.getNextCommand(command);
67             commandCenter.parse();
68         }
69     }
70
71     return (EXIT_SUCCESS);
72 }

```

It's perfectly acceptable for an application to make use of more than one `CommandParser` for different enum s, assuming they are listening on different ports.



## Chapter 17

# Parallel Processing

### 17.1 MPI Parallel Processing Package

The `MPI` package is a set of APIs used implement parallel processing using the `MPI` [12] network-based messaging system. The core concept implemented in the framework is that of a distributor, one or more receivers, work packages, and a processing element to be implemented by the application.

The classes that make up the `MPI` package encapsulate all the necessary function calls and error handling in order to create an `MPI` job. Furthermore, the distribution and reception of packages containing data to be used for processing are also encapsulated within the `MPI` Framework. Lastly, logging, both for the tracing of Framework activity as well as application needs, is managed by these classes.

Figure 17.1 on the following page shows the processes and data flow for a typical parallel job using components of the Framework. The distributor process executes code from the `Distributor` class, and the receiver process likewise executes `Receiver` class code. Within each process is shown the Framework packages that could be used for the job. The *Lib* element refers to the “black-box” component of software being tested, a fingerprint matching library, for example. In this example, a record store is used as the data source, and record keys are sent in the work packages. On the receiving side, the keys are used to read record data (values) from the same store.

On the receiving side of the job, the processing is separated into two areas of responsibility. Each `Task-N` is responsible for managing the workers (`Task-N:1 ... Task-N:c`) by starting them, accepting work requests, and sending a command to have them shut down when the job finishes. Each worker is responsible for consuming the contents of the work packages; that implementation is done in the application.

The partitioning of responsibility enables two features of the Framework. First, a worker process can handle signals or other errors and decide to shutdown without affecting the rest of the job. This capability is important when testing “black-box” software where function calls cannot be trusted.

Second, each `Task-N` can perform some work before creating the workers. One example is the loading of a large data set into memory; again, this is done within the application. Because `Task-N` calls the POSIX function `fork()` to create the workers, each worker inherits the work done by `Task-N`. In the case of a memory load, each worker now has that memory mapped into its address space. See Section 17.5 on page 57 for more details.

### 17.2 Work Package

A `WorkPackage` object wraps a simple container of data with some access methods. There is no information in this class pertaining to the nature or format of the data; it is simply treated as an array of unsigned integer values. However, clients of the class can store a value, the “number of elements”, that is transmitted along with the package. This value only has meaning to the client, and is usually equivalent to the number of larger-sized components making up the package. For example, this value may be the number of records contained in the package. It is up to the client of `WorkPackage` to understand how to separate the array into components.



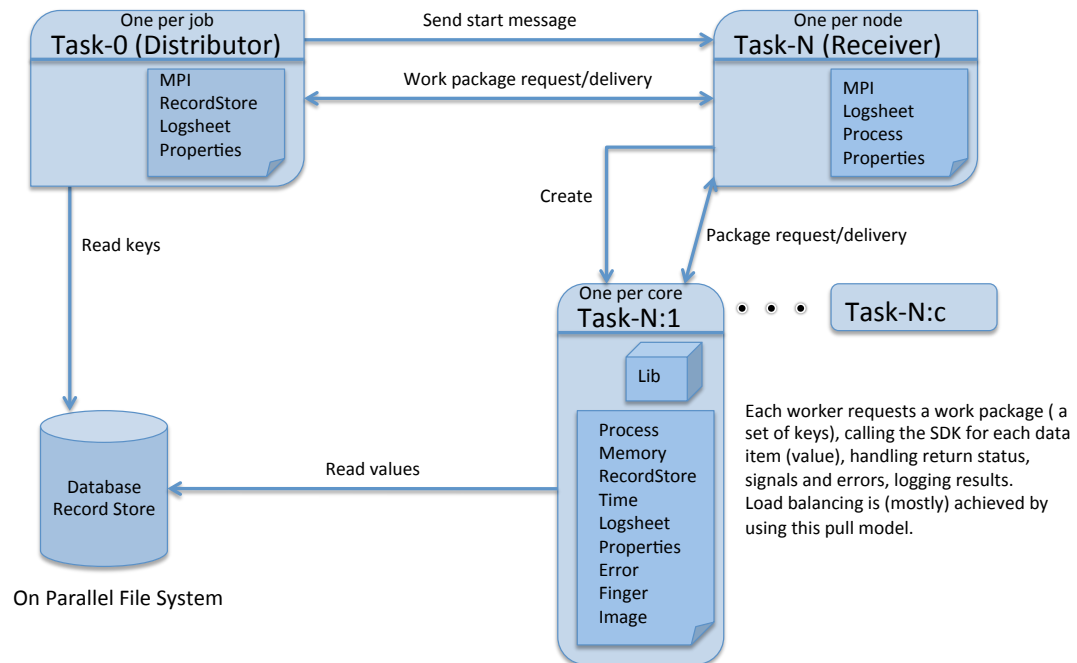


Figure 17.1: MPI Parallel Job Processes and Data Flow

The classes `RecordStoreDistributor` (Section 17.3.1) and `RecordProcessor` (Section 17.5.1 on the next page) are examples of `WorkPackage` clients that insert and remove data from a work package.

## 17.3 Distributor

The `Distributor` is an abstract class that encapsulates the MPI functionality and is responsible for distributing work packages to other elements within the MPI job (the receivers). However, this class is also responsible for coordinating the startup and shutdown of the receiver tasks. MPI messages are used for this coordination. An MPI job may fail to start if the distributor fails to initialize, or if none of the receivers initialize.

One method of the `Distributor` class, `createWorkPackage()`, is implemented by child classes. This method creates a single work package with the knowledge of how the elements of the package are to be stored in the package's data buffer. `RecordStoreDistributor` is an implementation of `Distributor`.

### 17.3.1 Record Store Distributor

`RecordStoreDistributor` reads records from a `RecordStore`, packs record keys, and optionally, values into a `WorkPackage`. This class inherits all of the MPI communication, intra-job coordination, logging, and other aspects of the `Distributor` parent class.

An application can create an instance of a `RecordStoreDistributor` with the name of a record store in order to distribute records for processing across the MPI job. Listing 17.3 on page 64 shows an example section of code to create a record store distributor. In this type of application there is no need for the application code to refine any of the Framework classes.

## 17.4 Receiver

The `Receiver` class encapsulates all the MPI messaging needed to participate in the MPI job as the receiver of data to be processed. In addition, this class is responsible for starting other processes that perform work on the actual data from the work package.

It is expected, as part of the MPI job, that a single receiver process will be started on each node in the job. More than one can be started, however. Each receiver starts one or more child processes to consume data. The receiver monitors each worker process and will instruct them to shut down when the job is finished (no more data), early termination signals are received, or in the case of errors encountered by the receiver.

By keeping the data consumers as separate processes, the receiving half of the MPI job can be more robust as a premature termination of a worker process (due to memory corruption, for example) will not affect other workers.

## 17.5 Work Package Processor

The `WorkPackageProcessor` class is pure-virtual and provides the interface for any class that uses a `WorkPackage` to receive data from the MPI Framework. `WorkPackageProcessor` also maintains a `Logsheet` object which can be used by subclasses to store log messages.

Implementations of this class can be considered to have dual responsibilities. First is the management of common state used by all workers (Task-N:c in Figure 17.1 on the preceding page); creating state data shared by all workers, for example. Second, as a factory to create a package consumer for the worker process.

The `performInitialization()` method is called before the `Receiver` object forks and creates the worker processes. The application can use this function to load a large data set into memory (taking advantage of copy-on-write memory semantics present in most modern operating systems), or perform any node-local setup that should only be done once the MPI job has begun.

`newProcessor()` returns a new instance of the package processor. This method is called by the Framework when a new process is started by the receiver to consume work packages sent by the distributor. This

method is a factory, creating new instances of the `WorkPackageProcessor` implementation. Therefore, it must create a “fully-formed” object that may have different state than that created by the class constructor. An example would be creating an output log file with record information. This output file would not be created in the constructor because the object returned from that will not process a work package; it is the factory object.

It is the responsibility of the `newProcessor()` method to ensure there is no resource contention between instances of this class, as the methods of this object will be executed within a separate process. The `MPI::generateUniqueID()` function can be used to create a name string that to identify the process.

### 17.5.1 Record Processor

`RecordProcessor` is a partial implementation of `WorkPackageProcessor` and defines the `processWorkPackage()` of the `WorkPackageProcessor` interface; other methods are declared as pure-virtual and must be implemented by a child class. In addition, `RecordProcessor` declares a new pure-virtual method, `processRecord()` to be implemented by a subclass to process a single record from the record store. In summary, `RecordProcessor` removes records from the work package to be processed within the subclass, which is defined by the application. See Listing 17.1 on the facing page and Listing 17.2 on page 60 for an example of such an implementation.

## 17.6 MPI Resources

Every MPI job depends on a set of properties contained within a text file. These properties are read into a `Properties` object contained within the `Resources` object.

The core MPI classes (`Distributor` and `Receiver`) use these properties:

**Workers Per Node** Used by the receiver process to start the required number of workers;

**Logsheet URL** Use by distributor and receiver processes (and children) to open the log.

The `Logsheet URL` property is optional, and if present all MPI Framework trace messages will be written to the specified logging target. Two types of Uniform Resource Locator schemes are allowed: `file://` and `syslog://`, corresponding to the types of `Logsheet` classes (Section 6.3 on page 16) in the Framework.

Subclasses and other components of the MPI Framework may add properties as needed, usually to the same file as the above properties. Record-based jobs (using `RecordStoreDistributor` and `RecordProcessor`), for example, have these additional properties:

**Input Record Store** The input record store;

**Chunk Size** How many record keys or key-value pairs to place into a work package.

For a record store job, an example properties file might be:

```
Input Record Store = test.rs
Chunk Size = 7
Workers Per Node = 3
Logsheet URL = file://mpi.log
```

Applications can add one or more properties to the file as needed. One example would be a URL for a `Logsheet` used only by the application.

## 17.7 MPI Runtime

The `Runtime` class is the interface between the application and the MPI runtime environment. The `argv` and `argc` parameters to the `main()` function as passed through to the `Runtime` object, then onto the core Open-MPI functions. The `Runtime` object also sets up a signal handler for the job, and starts the `Distributor`

and `Receiver` processes. A method is also provided for the application to abort the MPI job, providing for a somewhat clean shutdown.

One of the key features of an MPI job under the Framework is premature shutdown with minimal loss of work. Three types of exit condition can be set by sending a signal to the distributor, receiver or worker processes.

**SIGQUIT** Exit when the current work package is exhausted;

**SIGINT** Exit when the current work item is finished (“quick exit”);

**SIGTERM** Exit immediately (“term exit”).

For the normal exit and quick exit cases, a clean shutdown is performed for the distributor, receivers, and all worker processes. For term exit, each worker process is terminated immediately and therefore cannot finish processing the current work item. However, distributors and receivers will shutdown in a clean manner.

Any of the signals can be sent to the distributor process, which then sends messages to the receivers. In addition, if a signal is sent to a receiver, or worker process, only that process (receiver or worker) is affected, but the termination condition is communicated “up” the chain. By selectively sending signals to certain processes, a user can shutdown the entire job (send to the distributor), an entire node (send to the receiver on that node), or a single worker. A worker receiving a signal sends a message back to the receiver. Likewise, a receiver will communicate the shutdown state back to the distributor.

## 17.8 Logging

In order to aid tracing and debugging of a parallel job, the MPI Framework can be configured to write trace messages to the log storage. These trace messages are logged as debug messages instead of normal entries. The type and location of the log is given to the Framework by using a URL as a property when starting the MPI job (see Section 17.6 on the preceding page).

When the URL for a log is the `file://` type, the MPI Framework will create several log files on the node where it runs. The reason for this is that during `Receiver` processing, one or more worker processes are created in addition to the main receiver process. Each of these processes requires exclusive access to the file-based log sheet in order to avoid conflicts with the log entry commitment. The log files will be named with the property value as a prefix, and the hostname/MPI task number/process ID added as a suffix. For example, if the property is `file://mpijob.log`, a log file might have a name of `mpijob.log-node01-1-12345`.

To aid logging within the application, access to the `Logsheet` opened by the Framework is available via the class whose interface is implemented within the application, `WorkPackageProcessor`, for example.

Two wrapper functions, `MPI::logMessage()` and `MPI::logEntry()`, are provided in order to “safely” log. These functions handle all errors from the `Logsheet` object, and will turn off log message commitment once an error occurs. The Framework and application can continue processing.

## 17.9 MPI Framework Applications

An application of the MPI Framework is responsible for implementing several functions declared in the Framework, requiring subclassing of the MPI classes. In this section an example application that processes records from a store will be described.

Listing 17.1 shows the header file that declares a subclass of `RecordProcessor`. The `newProcessor()`, `performInitialization()`, and `processRecord()` methods are those required to complete an implementation of `RecordProcessor`. A memory buffer pointer is managed with a smart pointer object.

### Listing 17.1: MPI Framework Application Classes

```
1 | class TestRecordProcessor : public BiometricEvaluation::MPI::RecordProcessor {
```

```

2 public:
3     /**
4      * @brief
5      * The property string 'Logsheet URL'.
6      */
7     static const std::string RECORDLOGSHEETURLPROPERTY;
8
9     static const uint32_t SHAREDMEMORYSIZE = 2048;
10
11     TestRecordProcessor(
12         const std::string &propertiesFileName);
13     ~TestRecordProcessor();
14
15     std::shared_ptr<BE::MPI::WorkPackageProcessor>
16     newProcessor(std::shared_ptr<BE::IO::Logsheet> &logsheet);
17
18     void
19     performInitialization(std::shared_ptr<BE::IO::Logsheet> &logsheet);
20
21     void processRecord(const std::string &key);
22
23     void processRecord(
24         const std::string &key,
25         const BE::Memory::uint8Array &value);
26
27 protected:
28 private:
29     std::shared_ptr<BE::IO::Logsheet> _recordLogsheet;
30     std::shared_ptr<char> _sharedMemory;
31     uint32_t _sharedMemorySize;
32 };

```

Next, Listing 17.2 shows the implementation of the class methods. In this simple example, each record is acknowledged with a log entry.

Also shown in several of the methods is the use of the `Logsheet` object provided to the application by the Framework, along with wrapper functions, `logMessage()` and `logEntry()`.

The application also creates its own `Logsheet` object in order to separate Framework log messages from the application messages when processing the actual record. In error cases, the Framework log is used in order to keep the set of calls from the Framework to the application in sequence and package processing together.

A common memory buffer is allocated in `performInitialization()` method, and this buffer's pointer is copied to each processing instance in the `newProcessor()` method. Access to this common memory is shown in each `processRecord()` method. The actual memory buffer is not copied because the Framework will invoke the system call `fork()` which results in all memory of the parent process being copied into the child.

#### Listing 17.2: MPI Framework Application Implementation

```

1 #include <be_mpi_receiver.h>
2 #include <be_mpi_recordstoredistributor.h>
3 #include <be_mpi_runtime.h>
4
5 #include "test_be_mpi.h"
6
7 using namespace BiometricEvaluation;
8
9 static const std::string DefaultPropertiesFileName("test_be_mpi.props");

```

```

10 |
11 | /*
12 | * Implementations of the MPI RecordProcessor class interface.
13 | * Calls the parent constructor to manage the properties file name.
14 | */
15 | TestRecordProcessor::TestRecordProcessor(
16 |     const std::string &propertiesFileName) :
17 |     RecordProcessor(propertiesFileName)
18 | {
19 | }
20 |
21 | TestRecordProcessor::~TestRecordProcessor()
22 | {
23 | }
24 |
25 | /*
26 | * Factory object: Log our call and set up the shared memory buffer.
27 | */
28 | void
29 | TestRecordProcessor::performInitialization(
30 |     std::shared_ptr<IO::Logsheet> &logsheet)
31 | {
32 |     this->setLogsheet(logsheet);
33 |
34 |     /*
35 |     * Set up the memory that will be shared across all instances.
36 |     */
37 |     char *buf = (char *)malloc(SHAREDMEMORYSIZE);
38 |     strcpy(buf, "SHARED MEMORY");
39 |     this->_sharedMemorySize = SHAREDMEMORYSIZE;
40 |     this->_sharedMemory = std::unique_ptr<char>(buf);
41 |
42 |     *logsheet.get() << std::string(__FUNCTION__) << " called: ";
43 |     *logsheet.get()
44 |         << "Shared memory size is " << this->_sharedMemorySize
45 |         << " and contents is [" << buf << "];
46 |     BE::MPI::logEntry(*logsheet.get());
47 | }
48 |
49 | /*
50 | * Factory object: Create a new instance of the TestRecordProcess
51 | * that will work on work package records. Each instance gets
52 | * its own instance of the log sheet.
53 | */
54 | std::shared_ptr<BiometricEvaluation::MPI::WorkPackageProcessor>
55 | TestRecordProcessor::newProcessor(
56 |     std::shared_ptr<IO::Logsheet> &logsheet)
57 | {
58 |     std::string propertiesFileName =
59 |         this->getResources()->getPropertiesFileName();
60 |     TestRecordProcessor *processor =
61 |         new TestRecordProcessor(propertiesFileName);
62 |     processor->setLogsheet(logsheet);
63 |
64 |     /*
65 |     * If we have our own Logsheet property, and we can open

```

```

66      * that Logsheet, use it for record logging; otherwise,
67      * create a Null Logsheet for these events. We use the
68      * framework's Logsheet for tracing of processing, not
69      * record handling logs.
70      */
71      std::string url;
72      std::unique_ptr<BE::IO::PropertiesFile> props;
73      try {
74          /* It is crucial that the Properties file be
75           * opened read-only, else it will be rewritten
76           * when the unique ptr is destroyed, causing
77           * a race condition with other processes that
78           * are reading the file.
79           */
80          props.reset(new BE::IO::PropertiesFile(
81              propertiesFileName, IO::READONLY));
82          url = props->getProperty(
83              TestRecordProcessor::RECORDLOGSHEETURLPROPERTY);
84      } catch (BE::Error::Exception &e) {
85          url = "";
86      }
87      processor->_recordLogsheet = BE::MPI::openLogsheet(
88          url, "Test Record Processing");
89      processor->_sharedMemory = this->_sharedMemory;
90      processor->_sharedMemorySize = this->_sharedMemorySize;
91
92      std::shared_ptr<BiometricEvaluation::MPI::WorkPackageProcessor> sptr;
93      sptr.reset(processor);
94      return (sptr);
95  }
96
97  /*
98   * Helper function to log some information about a record.
99   */
100 static void
101 dumpRecord(
102     BE::IO::Logsheet &log,
103     const std::string key,
104     const Memory::uint8Array &val)
105 {
106     log << "Key [" << key << "]: ";
107     /* Dump some bytes from the record */
108     for (uint64_t i = 0; i < 8; i++) {
109         log << std::hex << (int)val[i] << " ";
110     }
111     log << " |";
112     for (uint64_t i = 0; i < 8; i++) {
113         log << (char)val[i];
114     }
115     log << "|";
116     BE::MPI::logEntry(log);
117 }
118
119 /*
120  * The worker object: Log to the Framework Logsheet, obtain the data for
121  * the record, and log some information to the record Logsheet.

```

```

122  */
123 void
124 TestRecordProcessor::processRecord(const std::string &key)
125 {
126     BE::IO::Logsheet *log = this->getLogsheet().get();
127
128     if (this->getResources()->haveRecordStore() == false) {
129         BE::MPI::logMessage(*log, "processRecord(" + key + ") "
130             + " called but have no record store; returning.");
131         return;
132     }
133     *log << "processRecord(" << key << ") called: ";
134     char *buf = this->_sharedMemory.get();
135     *log << "Shared memory size is " << this->_sharedMemorySize
136         << " and contents is [" << buf << "];";
137     BE::MPI::logEntry(*log);
138
139     Memory::uint8Array value(0);
140     std::shared_ptr<IO::RecordStore> inputRS =
141         this->getResources()->getRecordStore();
142     try {
143         inputRS->read(key, value);
144     } catch (Error::Exception &e) {
145         *log << string(__FUNCTION__) <<
146             " could not read record: " <<
147             e.whatString();
148         return;
149     }
150     /*
151     * Log record info to our own Logsheet instead of
152     * the one provided by the framework.
153     */
154     BE::IO::Logsheet *rlog = this->_recordLogsheet.get();
155     dumpRecord(*rlog, key, value);
156 }
157
158 /*
159 * The worker object: Log to the Framework Logsheet, and log some record
160 * information to the record Logsheet.
161 */
162 void
163 TestRecordProcessor::processRecord(
164     const std::string &key,
165     const BiometricEvaluation::Memory::uint8Array &value)
166 {
167     BE::IO::Logsheet *log = this->getLogsheet().get();
168     *log << "processRecord(" << key << ", [" << value << "] called: ";
169     char *buf = this->_sharedMemory.get();
170     *log << "Shared memory size is " << this->_sharedMemorySize
171         << " and contents is [" << buf << "];";
172     BE::MPI::logEntry(*log);
173
174     /*
175     * Log record info to our own Logsheet instead of
176     * the one provided by the framework.
177     */

```



```

178 |         BE::IO::Logsheet *rlog = this->_recordLogsheet.get();
179 |         dumpRecord(*rlog, key, value);
180 |     }

```

### Listing 17.3: MPI Framework Application Main

```

1 | int
2 | main(int argc, char* argv[])
3 | {
4 |     /*
5 |      * It is important that the MPI runtime environment be started
6 |      * prior to any other activity that may result in premature
7 |      * termination. Therefore, participate in the MPI environment, but
8 |      * don't create a Receiver or Distributor until any local items
9 |      * are take care of.
10 |     */
11 |     MPI::Runtime runtime(argc, argv);
12 |
13 |     std::unique_ptr<MPI::RecordStoreDistributor> distributor;
14 |     std::unique_ptr<MPI::Receiver> receiver;
15 |     std::shared_ptr<TestRecordProcessor> processor;
16 |
17 |     /*
18 |      * If there is any argument to the program, use keys only as the
19 |      * record distribution method. Otherwise, use keys and values.
20 |     */
21 |     bool includeValues;
22 |     if (argc == 1) {
23 |         MPI::printStatus("Test Distributor and Receiver, keys only");
24 |         includeValues = false;
25 |     } else {
26 |         MPI::printStatus("Test Distributor and Receiver, keys and values");
27 |         includeValues = true;
28 |     }
29 |     try {
30 |         distributor.reset(
31 |             new MPI::RecordStoreDistributor(propFile, includeValues));
32 |
33 |         processor.reset(new TestRecordProcessor(propFile));
34 |
35 |         receiver.reset(new MPI::Receiver(propFile, processor));
36 |
37 |         runtime.start(*distributor, *receiver);
38 |         runtime.shutdown();
39 |     } catch (Error::Exception &e) {
40 |         MPI::printStatus("Caught: " + e.whatString());
41 |         runtime.abort(EXIT_FAILURE);
42 |     } catch (...) {
43 |         MPI::printStatus("Caught some other exception");
44 |         runtime.abort(EXIT_FAILURE);
45 |     }
46 |
47 |     return (EXIT_SUCCESS);
48 | }

```

# References

- [1] *ANSI INCITS 378-2004: Finger Minutiae Format for Data Interchange*. ANSI/INCITS, 2004. 39, 43, 50
- [2] *ISO/IEC 19794-2: Information technology - Biometric data interchange formats - Part 2: Finger minutiae data*. ISO/IEC, first edition, 2005. 39, 50
- [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. 4, 39, 41, 47
- [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/>. 35
- [6] Independent JPEG Group. libjpeg, 2011. <http://www.ijg.org/>. 34
- [7] Joint Photographic Experts Group. JPEG2000 Image Standard, 1992. <http://www.jpeg.org/jpeg2000/index.html>. 34
- [8] Joint Photographic Experts Group. JPEG Image Standard, 2011. <http://www.jpeg.org/jpeg/index.html>. 34
- [9] International Committee for Information Technology Standards. <http://www.incits.org>. 43, 50
- [10] ISO/IEC Joint Technical Committee 1/SC 37 Biometrics. 43
- [11] Communications and Remote Sensing Lab, Université catholique de Louvain. OpenJPEG Library, 2011. <http://www.openjpeg.org/>. 34
- [12] Message Passing Interface Forum. <http://www.mpi-forum.org>. 4, 55
- [13] NIST Biometric Image Software, 2011. <http://www.nist.gov/itl/iad/ig/nbis.cfm>. 7, 34, 35
- [14] NIST Image Group. <http://www.nist.gov/itl/iad/ig>. 1
- [15] Greg Roelofs. libpng, 2011. <http://www.libpng.org/pub/png/libpng.html>. 35
- [16] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, special edition, 2000. 3, 9
- [17] The Syslog Protocol. <http://tools.ietf.org/html/rfc5424>. 18
- [18] Wavelet Scalar Quantization Gray-Scale Fingerprint Image Compression Standard, 2010. [https://www.fbibiospecs.org/docs/WSQ\\_Gray-scale\\_Specification\\_Version\\_3\\_1\\_Final.pdf](https://www.fbibiospecs.org/docs/WSQ_Gray-scale_Specification_Version_3_1_Final.pdf). 35



## **Appendix A**

### **API Reference**



## Appendix B

# Namespace Index

### B.1 Namespace List

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

<a href="#">BiometricEvaluation</a> . . . . .	81
<a href="#">BiometricEvaluation::Error</a> Exceptions, and other error handling . . . . .	82
<a href="#">BiometricEvaluation::Face</a> Biometric information relating to face images and derived information . . . . .	83
<a href="#">BiometricEvaluation::Feature</a> Definition of an MPEG4 <a href="#">Face</a> feature point. See ISO/IEC 14496-2 . . . . .	84
<a href="#">BiometricEvaluation::Finger</a> Biometric information relating to finger images and derived information . . . . .	86
<a href="#">BiometricEvaluation::Framework</a> Information about the framework . . . . .	88
<a href="#">BiometricEvaluation::Image</a> Basic information relating to images . . . . .	94
<a href="#">BiometricEvaluation::IO</a> Input/Output functionality . . . . .	96
<a href="#">BiometricEvaluation::IO::Utility</a> . . . . .	98
<a href="#">BiometricEvaluation::Iris</a> Biometric information relating to iris images and derived information . . . . .	105
<a href="#">BiometricEvaluation::Memory</a> Support for memory-related operations . . . . .	105
<a href="#">BiometricEvaluation::Memory::AutoArrayUtility</a> . . . . .	106
<a href="#">BiometricEvaluation::MPI</a> Common declarations and functions for the MPI-based functionality . . . . .	108
<a href="#">BiometricEvaluation::Process</a> Process information and controls . . . . .	110
<a href="#">BiometricEvaluation::System</a> Operating system, hardware, etc . . . . .	111
<a href="#">BiometricEvaluation::Text</a> Text processing for string objects . . . . .	112
<a href="#">BiometricEvaluation::Time</a> Support for time and timers . . . . .	115
<a href="#">BiometricEvaluation::View</a> View information . . . . .	117



# Appendix C

## Hierarchical Index

### C.1 Class Hierarchy

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

BiometricEvaluation::Finger::AN2KMinutiaeDataRecord . . . . .	124
BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric . . . . .	126
BiometricEvaluation::DataInterchange::AN2KRecord . . . . .	127
BiometricEvaluation::Memory::AutoArray< T > . . . . .	165
BiometricEvaluation::Memory::AutoArray< uint8_t > . . . . .	165
BiometricEvaluation::Memory::AutoBuffer< T > . . . . .	178
BiometricEvaluation::Memory::AutoBuffer< ANSL_NIST > . . . . .	178
BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet . . . . .	181
BiometricEvaluation::Process::CommandCenter< T, typename >::Command . . . . .	181
BiometricEvaluation::Process::CommandCenter< T, typename > . . . . .	182
BiometricEvaluation::Process::CommandCenter< T > . . . . .	182
BiometricEvaluation::Process::CommandParser< T > . . . . .	184
BiometricEvaluation::IO::Compressor . . . . .	192
BiometricEvaluation::IO::GZip . . . . .	237
BiometricEvaluation::Framework::ConstEnumMapWrapper< T > . . . . .	200
BiometricEvaluation::Image::Coordinate . . . . .	202
BiometricEvaluation::Feature::CorePoint . . . . .	203
BiometricEvaluation::Feature::DeltaPoint . . . . .	209
BiometricEvaluation::MPI::Distributor . . . . .	210
BiometricEvaluation::MPI::RecordStoreDistributor . . . . .	371
BiometricEvaluation::DataInterchange::AN2KRecord::DomainName . . . . .	211
BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry . . . . .	212
BiometricEvaluation::Framework::EnumerationFunctions< T > . . . . .	213
BiometricEvaluation::Framework::EnumMapWrapper< T > . . . . .	213
exception	
BiometricEvaluation::Error::Exception . . . . .	214
BiometricEvaluation::Error::ConversionError . . . . .	201
BiometricEvaluation::Error::DataError . . . . .	203
BiometricEvaluation::Error::FileError . . . . .	216
BiometricEvaluation::Error::MemoryError . . . . .	310
BiometricEvaluation::Error::NotImplemented . . . . .	323
BiometricEvaluation::Error::ObjectDoesNotExist . . . . .	324
BiometricEvaluation::Error::ObjectExists . . . . .	325



BiometricEvaluation::Error::ObjectIsClosed . . . . .	325
BiometricEvaluation::Error::ObjectIsOpen . . . . .	326
BiometricEvaluation::Error::ParameterError . . . . .	335
BiometricEvaluation::Error::StrategyError . . . . .	400
BiometricEvaluation::IO::FileLogCabinet . . . . .	217
BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem . . . . .	228
BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition . . . . .	229
BiometricEvaluation::Image::Image . . . . .	244
BiometricEvaluation::Image::BMP . . . . .	179
BiometricEvaluation::Image::JPEG . . . . .	284
BiometricEvaluation::Image::JPEG2000 . . . . .	286
BiometricEvaluation::Image::JPEGL . . . . .	288
BiometricEvaluation::Image::NetPBM . . . . .	318
BiometricEvaluation::Image::PNG . . . . .	336
BiometricEvaluation::Image::Raw . . . . .	351
BiometricEvaluation::Image::WSQ . . . . .	427
BiometricEvaluation::Memory::IndexedBuffer . . . . .	274
iterator	
BiometricEvaluation::IO::RecordStoreIterator . . . . .	372
BiometricEvaluation::Memory::AutoArrayIterator< B, T > . . . . .	172
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > . . . . .	330
BiometricEvaluation::Memory::OrderedMapIterator< Key, T > . . . . .	333
BiometricEvaluation::Process::Manager . . . . .	303
BiometricEvaluation::Process::ForkManager . . . . .	230
BiometricEvaluation::Process::POSIXThreadManager . . . . .	338
BiometricEvaluation::IO::ManifestEntry . . . . .	310
BiometricEvaluation::Process::MessageCenter . . . . .	311
BiometricEvaluation::MPI::MessageTag . . . . .	316
BiometricEvaluation::Feature::Minutiae . . . . .	317
BiometricEvaluation::Feature::AN2K7Minutiae . . . . .	121
BiometricEvaluation::Feature::INCITSMinutiae . . . . .	252
BiometricEvaluation::Feature::MinutiaPoint . . . . .	317
BiometricEvaluation::Feature::MPEGFacePoint . . . . .	318
BiometricEvaluation::Memory::OrderedMap< Key, T > . . . . .	327
BiometricEvaluation::Memory::OrderedMap< std::string, ManifestEntry > . . . . .	327
ostream	
BiometricEvaluation::IO::Logsheet . . . . .	295
BiometricEvaluation::IO::FileLogsheet . . . . .	218
BiometricEvaluation::IO::SysLogsheet . . . . .	400
BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification . . . . .	336
BiometricEvaluation::Face::PoseAngle . . . . .	338
BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate . . . . .	342
BiometricEvaluation::IO::Properties . . . . .	343
BiometricEvaluation::IO::PropertiesFile . . . . .	348
BiometricEvaluation::Iris::INCITSView::QualitySubBlock . . . . .	350
BiometricEvaluation::MPI::Receiver . . . . .	352
BiometricEvaluation::IO::RecordStore . . . . .	357
BiometricEvaluation::IO::ArchiveRecordStore . . . . .	158
BiometricEvaluation::IO::CompressedRecordStore . . . . .	186
BiometricEvaluation::IO::DBRecordStore . . . . .	204

BiometricEvaluation::IO::FileRecordStore . . . . .	223
BiometricEvaluation::IO::ListRecordStore . . . . .	289
BiometricEvaluation::IO::SQLiteRecordStore . . . . .	389
BiometricEvaluation::Image::Resolution . . . . .	377
BiometricEvaluation::MPI::Resources . . . . .	379
BiometricEvaluation::MPI::RecordStoreResources . . . . .	375
BiometricEvaluation::Feature::RidgeCountItem . . . . .	381
BiometricEvaluation::MPI::Runtime . . . . .	381
BiometricEvaluation::Process::Semaphore . . . . .	382
BiometricEvaluation::Error::SignalManager . . . . .	386
BiometricEvaluation::Image::Size . . . . .	389
BiometricEvaluation::Process::Statistics . . . . .	396
BiometricEvaluation::MPI::TaskCommand . . . . .	405
BiometricEvaluation::MPI::TaskStatus . . . . .	406
BiometricEvaluation::Time::Timer . . . . .	406
BiometricEvaluation::View::View . . . . .	407
BiometricEvaluation::Face::INCITSView . . . . .	254
BiometricEvaluation::Face::ISO2005View . . . . .	278
BiometricEvaluation::Finger::INCITSView . . . . .	266
BiometricEvaluation::Finger::ANSI2004View . . . . .	154
BiometricEvaluation::Finger::ANSI2007View . . . . .	156
BiometricEvaluation::Finger::ISO2005View . . . . .	280
BiometricEvaluation::Iris::INCITSView . . . . .	259
BiometricEvaluation::Iris::ISO2011View . . . . .	282
BiometricEvaluation::View::AN2KView . . . . .	136
BiometricEvaluation::Finger::AN2KView . . . . .	132
BiometricEvaluation::Finger::AN2KViewFixedResolution . . . . .	144
BiometricEvaluation::View::AN2KViewVariableResolution . . . . .	151
BiometricEvaluation::Finger::AN2KViewVariableResolution . . . . .	147
BiometricEvaluation::Finger::AN2KViewCapture . . . . .	139
BiometricEvaluation::Finger::AN2KViewLatent . . . . .	145
BiometricEvaluation::Time::Watchdog . . . . .	411
BiometricEvaluation::Process::Worker . . . . .	415
BiometricEvaluation::Process::MessageCenterListener . . . . .	314
BiometricEvaluation::Process::MessageCenterReceiver . . . . .	315
BiometricEvaluation::Process::WorkerController . . . . .	420
BiometricEvaluation::Process::ForkWorkerController . . . . .	234
BiometricEvaluation::Process::POSIXThreadWorkerController . . . . .	340
BiometricEvaluation::MPI::WorkPackage . . . . .	424
BiometricEvaluation::MPI::WorkPackageProcessor . . . . .	425
BiometricEvaluation::MPI::RecordProcessor . . . . .	354



# Appendix D

## Class Index

### D.1 Class List

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

<a href="#">BiometricEvaluation::Feature::AN2K7Minutiae</a>	
A class to represent a set of minutiae in an ANSI/NIST record . . . . .	121
<a href="#">BiometricEvaluation::Finger::AN2KMinutiaeDataRecord</a>	
Representation of a Type-9 Record from an AN2K file . . . . .	124
<a href="#">BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric</a>	
A structure to represent an AN2K quality metric . . . . .	126
<a href="#">BiometricEvaluation::DataInterchange::AN2KRecord</a>	
A class to represent an entire ANSI/NIST record . . . . .	127
<a href="#">BiometricEvaluation::Finger::AN2KView</a>	
A class to represent single finger view and derived information . . . . .	132
<a href="#">BiometricEvaluation::View::AN2KView</a>	
A class to represent single biometric view and derived information . . . . .	136
<a href="#">BiometricEvaluation::Finger::AN2KViewCapture</a>	
Represents an ANSI/NIST variable-resolution finger image . . . . .	139
<a href="#">BiometricEvaluation::Finger::AN2KViewFixedResolution</a>	
A class to represent single finger view and derived information . . . . .	144
<a href="#">BiometricEvaluation::Finger::AN2KViewLatent</a>	
. . . . .	145
<a href="#">BiometricEvaluation::Finger::AN2KViewVariableResolution</a>	
A class to represent single finger view based on an ANSI/NIST record . . . . .	147
<a href="#">BiometricEvaluation::View::AN2KViewVariableResolution</a>	
A class to represent single view based on an ANSI/NIST record . . . . .	151
<a href="#">BiometricEvaluation::Finger::ANSI2004View</a>	
A class to represent single finger view and derived information . . . . .	154
<a href="#">BiometricEvaluation::Finger::ANSI2007View</a>	
A class to represent single finger view and derived information . . . . .	156
<a href="#">BiometricEvaluation::IO::ArchiveRecordStore</a>	
This class implements the <a href="#">IO::RecordStore</a> interface by storing data items in single file, with an associated manifest file . . . . .	158
<a href="#">BiometricEvaluation::Memory::AutoArray&lt; T &gt;</a>	
A C-style array wrapped in the facade of a C++ STL container . . . . .	165
<a href="#">BiometricEvaluation::Memory::AutoArrayIterator&lt; B, T &gt;</a>	
RandomAccessIterator for any <a href="#">AutoArray</a> . . . . .	172
<a href="#">BiometricEvaluation::Memory::AutoBuffer&lt; T &gt;</a>	
. . . . .	178

<a href="#">BiometricEvaluation::Image::BMP</a>	
A BMP-encoded image . . . . .	179
<a href="#">BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet</a>	181
<a href="#">BiometricEvaluation::Process::CommandCenter&lt; T, typename &gt;::Command</a>	181
<a href="#">BiometricEvaluation::Process::CommandCenter&lt; T, typename &gt;</a>	182
<a href="#">BiometricEvaluation::Process::CommandParser&lt; T &gt;</a>	184
<a href="#">BiometricEvaluation::IO::CompressedRecordStore</a>	
Sibling-implemented <a href="#">RecordStore</a> with Compression . . . . .	186
<a href="#">BiometricEvaluation::IO::Compressor</a>	192
<a href="#">BiometricEvaluation::Framework::ConstEnumMapWrapper&lt; T &gt;</a>	
Wrapper class around an individual enumeration entity (const) . . . . .	200
<a href="#">BiometricEvaluation::Error::ConversionError</a>	
Error when converting one object into another, a property value from string to int, for example . . . . .	201
<a href="#">BiometricEvaluation::Image::Coordinate</a>	
A structure to contain a two-dimensional coordinate without a specified origin . . . . .	202
<a href="#">BiometricEvaluation::Feature::CorePoint</a>	
Representation of the core . . . . .	203
<a href="#">BiometricEvaluation::Error::DataError</a>	
Error when reading data from an external source . . . . .	203
<a href="#">BiometricEvaluation::IO::DBRecordStore</a>	
A class that implements <a href="#">IO::RecordStore</a> using a Berkeley DB database as the underlying record storage system . . . . .	204
<a href="#">BiometricEvaluation::Feature::DeltaPoint</a>	
Representation of the delta . . . . .	209
<a href="#">BiometricEvaluation::MPI::Distributor</a>	
A class to represent an <a href="#">MPI</a> task that distributes work to other tasks . . . . .	210
<a href="#">BiometricEvaluation::DataInterchange::AN2KRecord::DomainName</a>	
Representation of a domain name for the user-defined Type-2 logical record implementation . . . . .	211
<a href="#">BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry</a>	212
<a href="#">BiometricEvaluation::Framework::EnumerationFunctions&lt; T &gt;</a>	213
<a href="#">BiometricEvaluation::Framework::EnumMapWrapper&lt; T &gt;</a>	
Wrapper class around an individual enumeration entity (non-const) . . . . .	213
<a href="#">BiometricEvaluation::Error::Exception</a>	
The parent class of all <a href="#">BiometricEvaluation</a> exceptions . . . . .	214
<a href="#">BiometricEvaluation::Error::FileError</a>	
File error when opening, reading, writing, etc . . . . .	216
<a href="#">BiometricEvaluation::IO::FileLogCabinet</a>	217
<a href="#">BiometricEvaluation::IO::FileLogsheets</a>	
A class to represent a single logging mechanism with a file as the backing store . . . . .	218
<a href="#">BiometricEvaluation::IO::FileRecordStore</a>	223
<a href="#">BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem</a>	
Representation of information about a fingerprint reader system . . . . .	228
<a href="#">BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition</a>	
Locations of an individual finger segment in a slap . . . . .	229
<a href="#">BiometricEvaluation::Process::ForkManager</a>	
Manager implementation that starts Workers by calling fork(2) . . . . .	230
<a href="#">BiometricEvaluation::Process::ForkWorkerController</a>	
Wrapper of a <a href="#">Worker</a> returned from a <a href="#">Process::ForkManager</a> . . . . .	234
<a href="#">BiometricEvaluation::IO::GZip</a>	
Compressor for gzip compression from zlib . . . . .	237

<a href="#">BiometricEvaluation::Image::Image</a>	
Represent attributes common to all images . . . . .	244
<a href="#">BiometricEvaluation::Feature::INCITSMinutiae</a>	
A class to represent a set of minutiae in an ANSI/INCITS record . . . . .	252
<a href="#">BiometricEvaluation::Face::INCITSView</a>	
A class to represent single facial image view and derived information . . . . .	254
<a href="#">BiometricEvaluation::Iris::INCITSView</a>	
A class to represent single iris view and derived information . . . . .	259
<a href="#">BiometricEvaluation::Finger::INCITSView</a>	
A class to represent single finger view and derived information . . . . .	266
<a href="#">BiometricEvaluation::Memory::IndexedBuffer</a>	
Manage a memory buffer with an index . . . . .	274
<a href="#">BiometricEvaluation::Face::ISO2005View</a>	
A class to represent single face view and derived information . . . . .	278
<a href="#">BiometricEvaluation::Finger::ISO2005View</a>	
A class to represent single finger view and derived information . . . . .	280
<a href="#">BiometricEvaluation::Iris::ISO2011View</a>	
A class to represent single iris view and derived information . . . . .	282
<a href="#">BiometricEvaluation::Image::JPEG</a>	
A JPEG-encoded image . . . . .	284
<a href="#">BiometricEvaluation::Image::JPEG2000</a>	
A JPEG-2000-encoded image . . . . .	286
<a href="#">BiometricEvaluation::Image::JPEGL</a>	
A Lossless JPEG-encoded image . . . . .	288
<a href="#">BiometricEvaluation::IO::ListRecordStore</a>	
<a href="#">RecordStore</a> that reads a list of keys from a text file, and retrieves the data from another	
<a href="#">RecordStore</a> . . . . .	289
<a href="#">BiometricEvaluation::IO::Logsheet</a>	
A class to represent a logging mechanism . . . . .	295
<a href="#">BiometricEvaluation::Process::Manager</a>	
An interface for intranode process management classes . . . . .	303
<a href="#">BiometricEvaluation::IO::ManifestEntry</a>	
. . . . .	310
<a href="#">BiometricEvaluation::Error::MemoryError</a>	
An error occurred when allocating an object . . . . .	310
<a href="#">BiometricEvaluation::Process::MessageCenter</a>	
. . . . .	311
<a href="#">BiometricEvaluation::Process::MessageCenterListener</a>	
. . . . .	314
<a href="#">BiometricEvaluation::Process::MessageCenterReceiver</a>	
Receives message from a client, forwarding to the central <a href="#">MessageCenter</a> . . . . .	315
<a href="#">BiometricEvaluation::MPI::MessageTag</a>	
The types of messages sent between <a href="#">MPI</a> task processes . . . . .	316
<a href="#">BiometricEvaluation::Feature::Minutiae</a>	
A class to represent a set of minutiae data points . . . . .	317
<a href="#">BiometricEvaluation::Feature::MinutiaPoint</a>	
Representation of a finger minutiae data point . . . . .	317
<a href="#">BiometricEvaluation::Feature::MPEGFacePoint</a>	
Representation of a feature point and a set of points . . . . .	318
<a href="#">BiometricEvaluation::Image::NetPBM</a>	
A NetPBM-encoded image . . . . .	318
<a href="#">BiometricEvaluation::Error::NotImplemented</a>	
A <a href="#">NotImplemented</a> object is thrown when the underlying implementation of this interface	
has not or could not be created . . . . .	323

<a href="#">BiometricEvaluation::Error::ObjectDoesNotExist</a>	
The named object does not exist . . . . .	324
<a href="#">BiometricEvaluation::Error::ObjectExists</a>	
The named object exists and will not be replaced . . . . .	325
<a href="#">BiometricEvaluation::Error::ObjectIsClosed</a>	
The object is closed . . . . .	325
<a href="#">BiometricEvaluation::Error::ObjectIsOpen</a>	
The object is already opened . . . . .	326
<a href="#">BiometricEvaluation::Memory::OrderedMap&lt; Key, T &gt;</a>	327
<a href="#">BiometricEvaluation::Memory::OrderedMapConstIterator&lt; Key, T &gt;</a>	330
<a href="#">BiometricEvaluation::Memory::OrderedMapIterator&lt; Key, T &gt;</a>	333
<a href="#">BiometricEvaluation::Error::ParameterError</a>	
An invalid parameter was passed to a constructor or method . . . . .	335
<a href="#">BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification</a>	
Pattern classification codes . . . . .	336
<a href="#">BiometricEvaluation::Image::PNG</a>	
A PNG-encoded image . . . . .	336
<a href="#">BiometricEvaluation::Face::PoseAngle</a>	
Representation of pose angle and uncertainty . . . . .	338
<a href="#">BiometricEvaluation::Process::POSIXThreadManager</a>	
Manager implementation that starts Workers in POSIX threads . . . . .	338
<a href="#">BiometricEvaluation::Process::POSIXThreadWorkerController</a>	
Decorated <a href="#">Worker</a> returned from a <a href="#">Process::POSIXThreadManager</a> . . . . .	340
<a href="#">BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate</a>	
Offsets to the bounding boxes for the EJI, full finger views, or EJI segments . . . . .	342
<a href="#">BiometricEvaluation::IO::Properties</a>	
Maintain key/value pairs of strings, with each property matched to one value . . . . .	343
<a href="#">BiometricEvaluation::IO::PropertiesFile</a>	
A <a href="#">Properties</a> object persisted in an file on disk . . . . .	348
<a href="#">BiometricEvaluation::Iris::INCITSView::QualitySubBlock</a>	
Representation of an iris quality block . . . . .	350
<a href="#">BiometricEvaluation::Image::Raw</a>	
An image with no encoding or compression . . . . .	351
<a href="#">BiometricEvaluation::MPI::Receiver</a>	
A class to represent an <a href="#">MPI</a> task that receives <a href="#">WorkPackages</a> containers from the <a href="#">Distributor</a> . . . . .	352
<a href="#">BiometricEvaluation::MPI::RecordProcessor</a>	
An implementation of a work package processor that will extract record store keys, and optionally, values, from a <a href="#">WorkPackage</a> . . . . .	354
<a href="#">BiometricEvaluation::IO::RecordStore</a>	
A class to represent a data storage mechanism . . . . .	357
<a href="#">BiometricEvaluation::MPI::RecordStoreDistributor</a>	
An implementation of the <a href="#">Distributor</a> abstraction that uses a record store for input to create the work packages . . . . .	371
<a href="#">BiometricEvaluation::IO::RecordStoreIterator</a>	
Generic <a href="#">ForwardIterator</a> for all <a href="#">RecordStores</a> . . . . .	372
<a href="#">BiometricEvaluation::MPI::RecordStoreResources</a>	
A class to represent a set of resources needed by an <a href="#">MPI</a> program using a <a href="#">RecordStore</a> for input . . . . .	375
<a href="#">BiometricEvaluation::Image::Resolution</a>	
A structure to represent the resolution of an image . . . . .	377
<a href="#">BiometricEvaluation::MPI::Resources</a>	379

<a href="#">BiometricEvaluation::Feature::RidgeCountItem</a>	
Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number . . . . .	381
<a href="#">BiometricEvaluation::MPI::Runtime</a>	
Runtime support for the startup/shutdown of <a href="#">MPI</a> jobs . . . . .	381
<a href="#">BiometricEvaluation::Process::Semaphore</a>	
Represent a semaphore that can be used for interprocess communication . . . . .	382
<a href="#">BiometricEvaluation::Error::SignalManager</a>	
A <a href="#">SignalManager</a> object is used to handle signals that come from the operating system . .	386
<a href="#">BiometricEvaluation::Image::Size</a>	
A structure to represent the size of an image, in pixels . . . . .	389
<a href="#">BiometricEvaluation::IO::SQLiteRecordStore</a>	
A <a href="#">RecordStore</a> implementation using a SQLite database as the underlying record storage system . . . . .	389
<a href="#">BiometricEvaluation::Process::Statistics</a>	
Interface for gathering process statistics, such as memory usage, system time, etc . . . . .	396
<a href="#">BiometricEvaluation::Error::StrategyError</a>	
A <a href="#">StrategyError</a> object is thrown when the underlying implementation of this interface encounters an error . . . . .	400
<a href="#">BiometricEvaluation::IO::SysLogsheet</a>	
A class to represent a single logging mechanism to a logging service on the network . . .	400
<a href="#">BiometricEvaluation::MPI::TaskCommand</a>	
The command given to an <a href="#">MPI</a> task . . . . .	405
<a href="#">BiometricEvaluation::MPI::TaskStatus</a>	
The status of an <a href="#">MPI</a> distributor or receiver task . . . . .	406
<a href="#">BiometricEvaluation::Time::Timer</a>	
This class can be used by applications to report the amount of time a block of code takes to execute . . . . .	406
<a href="#">BiometricEvaluation::View::View</a>	
A class to represent single biometric element view . . . . .	407
<a href="#">BiometricEvaluation::Time::Watchdog</a>	
A <a href="#">Watchdog</a> object can be used by applications to limit the amount of processing time taken by a block of code . . . . .	411
<a href="#">BiometricEvaluation::Process::Worker</a>	
An abstraction of an instance that performs work on given data . . . . .	415
<a href="#">BiometricEvaluation::Process::WorkerController</a>	
Wrapper of a <a href="#">Worker</a> returned from a <a href="#">Process::Manager</a> . . . . .	420
<a href="#">BiometricEvaluation::MPI::WorkPackage</a>	
A class to represent a piece of work to be acted upon by a processor . . . . .	424
<a href="#">BiometricEvaluation::MPI::WorkPackageProcessor</a>	
Represents an object that processes the contents of a work package . . . . .	425
<a href="#">BiometricEvaluation::Image::WSQ</a>	
A WSQ-encoded image . . . . .	427





# Appendix E

## Namespace Documentation

### E.1 BiometricEvaluation Namespace Reference

#### Namespaces

- [Error](#)  
*Exceptions, and other error handling.*
- [Face](#)  
*Biometric information relating to face images and derived information.*
- [Feature](#)  
*Definition of an MPEG4 [Face](#) feature point. See ISO/IEC 14496-2.*
- [Finger](#)  
*Biometric information relating to finger images and derived information.*
- [Framework](#)  
*Information about the framework.*
- [Image](#)  
*Basic information relating to images.*
- [IO](#)  
*Input/Output functionality.*
- [Iris](#)  
*Biometric information relating to iris images and derived information.*
- [Memory](#)  
*Support for memory-related operations.*
- [MPI](#)  
*Common declarations and functions for the MPI-based functionality.*
- [Process](#)  
*[Process](#) information and controls.*
- [System](#)  
*Operating system, hardware, etc.*
- [Text](#)  
*[Text](#) processing for string objects.*
- [Time](#)  
*Support for time and timers.*
- [View](#)  
*[View](#) information.*

### E.1.1 Detailed Description

This software was developed at the National Institute of Standards and Technology (NIST) by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code, this software is not subject to copyright protection and is in the public domain. NIST assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic.

This software was developed at the National Institute of Standards and Technology (NIST) by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code, this software is not subject to copyright protection and is in the public domain. NIST assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. An interface to the object that processes a package of work from the [MPI Receiver](#).

## E.2 BiometricEvaluation::Error Namespace Reference

Exceptions, and other error handling.

### Classes

- 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 [Exception](#)  
*The parent class of all [BiometricEvaluation](#) exceptions.*
- class [FileError](#)  
*File error when opening, reading, writing, etc.*
- class [MemoryError](#)  
*An error occurred when allocating an object.*
- class [NotImplemented](#)  
*A [NotImplemented](#) object is thrown when the underlying implementation of this interface has not or could not be created.*
- class [ObjectDoesNotExist](#)  
*The named object does not exist.*
- class [ObjectExists](#)  
*The named object exists and will not be replaced.*
- class [ObjectIsClosed](#)  
*The object is closed.*
- class [ObjectIsOpen](#)  
*The object is already opened.*
- class [ParameterError](#)  
*An invalid parameter was passed to a constructor or method.*
- class [SignalManager](#)  
*A [SignalManager](#) object is used to handle signals that come from the operating system.*
- class [StrategyError](#)  
*A [StrategyError](#) object is thrown when the underlying implementation of this interface encounters an error.*

## Functions

- std::string [errorStr](#) (bool includeErrno=false)  
*Convert the value of errno to a human-readable error message.*
- void **SignalManagerSighandler** (int signo, siginfo\_t \*info, void \*uap)

### E.2.1 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.

### E.2.2 Function Documentation

std::string BiometricEvaluation::Error::errorStr ( bool *includeErrno* = *false* )

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

Parameters

<i>includeErrno</i>	Whether or not to include the value of errno in the returned string.
---------------------	--

Returns

The current error message specified by errno.

## E.3 BiometricEvaluation::Face Namespace Reference

Biometric information relating to face images and derived information.

### Classes

- class [INCITSView](#)  
*A class to represent single facial image view and derived information.*
- class [ISO2005View](#)  
*A class to represent single face view and derived information.*
- struct [PoseAngle](#)  
*Representation of pose angle and uncertainty.*

### Typedefs

- typedef std::vector  
< [BiometricEvaluation::Face::Property](#) > [PropertySet](#)

### Enumerations

- enum [Gender](#) { **Unspecified** = 0x00, **Male** = 0x01, **Female** = 0x02, **Unknown** = 0xFF }  
*Gender identifiers.*
- enum [EyeColor](#) {  
  **Unspecified** = 0x00, **Black** = 0x01, **Blue** = 0x02, **Brown** = 0x03,  
  **Gray** = 0x04, **Green** = 0x05, **MultiColored** = 0x06, **Pink** = 0x07,  
  **Unknown** = 0xFF }

- Eye color.*
- enum [HairColor](#) {  
**Unspecified** = 0x00, **Bald** = 0x01, **Black** = 0x02, **Blonde** = 0x03,  
**Brown** = 0x04, **Gray** = 0x05, **White** = 0x06, **Red** = 0x07,  
**Unknown** = 0xFF }
- Hair color.*
- enum [Property](#) {  
**Glasses** = 1, **Moustache** = 2, **Beard** = 3, **Teeth** = 4,  
**Blink** = 5, **MouthOpen** = 6, **LeftEyePatch** = 7, **RightEyePatch** = 8,  
**DarkGlasses** = 9, **MedicalCondition** = 10 }
- Face property codes.*
- enum [Expression](#) {  
**Unspecified** = 0x0000, **Neutral** = 0x0001, **SmileClosedJaw** = 0x0002, **SmileOpenJaw** = 0x0003,  
**RaisedEyebrows** = 0x0004, **EyesLookingAway** = 0x0005, **Squinting** = 0x0006, **Frowning** = 0x0007 }
- Face expression codes.*
- enum [ImageType](#) { **Basic** = 0x00, **FullFrontal** = 0x01, **TokenFrontal** = 0x02 }
- Face image type classification codes.*
- enum [ImageDataType](#) { **JPEG** = 0x00, **JPEG2000** = 0x01 }
- Face image data type classification codes.*
- enum [ColorSpace](#) {  
**Unspecified** = 0x00, **RGB24** = 0x01, **YUV422** = 0x02, **Grayscale8** = 0x03,  
**Other** = 0x04 }
- Color space codes.*
- enum [SourceType](#) {  
**Unspecified** = 0x00, **StaticPhotoUnknown** = 0x01, **StaticPhotoDigitalStill** = 0x02, **StaticPhotoScan** = 0x03,  
**VideoFrameUnknown** = 0x04, **VideoFrameAnalog** = 0x05, **VideoFrameDigital** = 0x06, **Unknown** = 0x07 }
- Source type codes.*

### E.3.1 Detailed Description

Biometric information relating to face images and derived information.

The [Face](#) package gathers all face related matters, including classes to represent face information and helper functions for conversion between biometric representations. Contained within this namespace are classes to represent specific record formats, such as ISO 19794-5.

### E.3.2 Typedef Documentation

`typedef std::vector<BiometricEvaluation::Face::Property> BiometricEvaluation::Face::PropertySet`

A set of properties.

## E.4 BiometricEvaluation::Feature Namespace Reference

Definition of an MPEG4 [Face](#) feature point. See ISO/IEC 14496-2.

## Classes

- class [AN2K7Minutiae](#)  
*A class to represent a set of minutiae in an ANSI/NIST record.*
- struct [CorePoint](#)  
*Representation of the core.*
- struct [DeltaPoint](#)  
*Representation of the delta.*
- class [INCITSMinutiae](#)  
*A class to represent a set of minutiae in an ANSI/INCITS record.*
- class [Minutiae](#)  
*A class to represent a set of minutiae data points.*
- struct [MinutiaPoint](#)  
*Representation of a finger minutiae data point.*
- struct [MPEGFacePoint](#)  
*Representation of a feature point and a set of points.*
- struct [RidgeCountItem](#)  
*Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number.*

## Typedefs

- using [AN2K7MinutiaeSet](#) = std::vector< std::shared\_ptr< [AN2K7Minutiae](#) >>
- using [MinutiaPoint](#) = struct [MinutiaPoint](#)
- using [MinutiaPointSet](#) = std::vector< [MinutiaPoint](#) >
- using [RidgeCountItem](#) = struct [RidgeCountItem](#)
- using [RidgeCountItemSet](#) = std::vector< [RidgeCountItem](#) >
- using [CorePoint](#) = struct [CorePoint](#)
- using [CorePointSet](#) = std::vector< [CorePoint](#) >
- using [DeltaPoint](#) = struct [DeltaPoint](#)
- using [DeltaPointSet](#) = std::vector< [DeltaPoint](#) >
- using [MinutiaeSet](#) = std::vector< std::shared\_ptr< [Minutiae](#) >>
- typedef std::vector< [MPEGFacePoint](#) > [MPEGFacePointSet](#)

## Enumerations

- enum [MinutiaeFormat](#) {  
  **AN2K7** = 0, **IAFIS**, **Cogent**, **Motorola**,  
  **Sagem**, **NEC**, **Identix**, **M1** }  
*Enumerate the minutiae format standards.*
- enum [MinutiaeType](#) { **RidgeEnding** = 0, **Bifurcation**, **Compound**, **Other** }  
*Enumerate the types of minutiae: Ridge Ending, Bifurcation, Compound, or other.*
- enum [RidgeCountExtractionMethod](#) { **NonSpecific** = 0, **FourNeighbor** = 1, **EightNeighbor** = 2, **Other** = 3 }  
*Enumerate the types of extraction methods for ridge counts.*

## Functions

- `std::ostream & operator<< (std::ostream &, const AN2K7Minutiae::FingerprintReadingSystem &)`  
*Output stream overload for FingerprintReadingSystem.*
- `std::ostream & operator<< (std::ostream &, const MinutiaPoint &)`
- `std::ostream & operator<< (std::ostream &, const RidgeCountItem &)`
- `std::ostream & operator<< (std::ostream &, const CorePoint &)`
- `std::ostream & operator<< (std::ostream &, const DeltaPoint &)`

### E.4.1 Detailed Description

Definition of an MPEG4 [Face](#) feature point. See ISO/IEC 14496-2.

## E.5 BiometricEvaluation::Finger Namespace Reference

Biometric information relating to finger images and derived information.

### Classes

- 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

- using **PositionSet** = `std::vector< Position >`
- using **PositionDescriptors** = `std::map< Position, FingerImageCode >`

## Enumerations

- enum [PatternClassification](#) {  
**PlainArch** = 0, **TentedArch**, **RadialLoop**, **UlnarLoop**,  
**PlainWhorl**, **CentralPocketLoop**, **DoubleLoop**, **AccidentalWhorl**,  
**Whorl**, **RightSlantLoop**, **LeftSlantLoop**, **Scar**,  
**Amputation**, **Unknown** }
  - enum [Position](#) {  
**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 }
- Finger position codes.*
- enum [Impression](#) {  
**LiveScanPlain** = 0, **LiveScanRolled**, **NonLiveScanPlain**, **NonLiveScanRolled**,  
**LatentImpression**, **LatentTracing**, **LatentPhoto**, **LatentLift**,  
**LiveScanVerticalSwipe**, **LiveScanPalm**, **NonLiveScanPalm**, **LatentPalmImpression**,  
**LatentPalmTracing**, **LatentPalmPhoto**, **LatentPalmLift**, **LiveScanOpticalContactPlain**,  
**LiveScanOpticalContactRolled**, **LiveScanNonOpticalContactPlain**, **LiveScanNonOpticalContact**↵  
**Rolled**, **LiveScanOpticalContactlessPlain**,  
**LiveScanOpticalContactlessRolled**, **LiveScanNonOpticalContactlessPlain**, **LiveScanNonOptical**↵  
**ContactlessRolled**, **Other**,  
**Unknown** }
  - enum [FingerImageCode](#) {  
**EJI** = 0, **RolledTip**, **FullFingerRolled**, **FullFingerPlainLeft**,  
**FullFingerPlainCenter**, **FullFingerPlainRight**, **ProximalSegment**, **DistalSegment**,  
**MedialSegment**, **NA** }

## Functions

- std::ostream & [operator<<](#) (std::ostream &stream, const [AN2KViewCapture::FingerSegmentPosition](#) &fsp)  
*Output stream overload for FingerSegmentPosition.*
- std::ostream & [operator<<](#) (std::ostream &stream, const [AN2KViewVariableResolution::PrintPosition](#)↵  
[Coordinate](#) &ppc)  
*Output stream overload for PrintPositionCoordinate.*

### E.5.1 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.

### E.5.2 Enumeration Type Documentation

**enum BiometricEvaluation::Finger::FingerImageCode** [strong]

Joint and tip codes.



**enum BiometricEvaluation::Finger::Impression [strong]**

[Finger](#) and palm impression types.

**enum BiometricEvaluation::Finger::PatternClassification [strong]**

Pattern classification codes.

**enum BiometricEvaluation::Finger::Position [strong]**

[Finger](#) position codes.

These codes match those in ANSI/NIST. Other minutiae formats may have to map codes into this set.

### E.5.3 Function Documentation

**std::ostream& BiometricEvaluation::Finger::operator<< ( std::ostream & *stream*, const AN2KViewVariableResolution::PrintPositionCoordinate & *ppc* )**

Output stream overload for PrintPositionCoordinate.

Parameters

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

Returns

Stream with a ppc textual representation appended.

## E.6 BiometricEvaluation::Framework Namespace Reference

Information about the framework.

### Classes

- class [ConstEnumMapWrapper](#)  
*Wrapper class around an individual enumeration entity (const).*
- class [EnumerationFunctions](#)
- class [EnumMapWrapper](#)  
*Wrapper class around an individual enumeration entity (non-const).*

### 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 getVersion ()`  
Version string of compiler used to compile this framework.
  - `template<typename T > bool operator== (const std::string &lhs, const EnumMapWrapper< T > &rhs)`  
Determine if a string and the string representation of an enumeration are equal.
  - `template<typename T > bool operator== (const EnumMapWrapper< T > &lhs, const std::string &rhs)`  
Determine if a string representation of an enumeration and a string are equal.
  - `template<typename T > bool operator!= (const std::string &lhs, const EnumMapWrapper< T > &rhs)`  
Determine if a string and the string representation of an enumeration are not equal.
  - `template<typename T > bool operator!= (const EnumMapWrapper< T > &lhs, const std::string &rhs)`  
Determine if a string representation of an enumeration and a string are not equal.
  - `template<typename T > std::ostream & operator<< (std::ostream &stream, const EnumMapWrapper< T > &kind)`  
Append the string representation of an enumeration into a stream.
  - `template<typename T > std::string operator+ (const std::string &lhs, const Framework::EnumMapWrapper< T > &rhs)`  
Concatenate the string representation of an enumeration to an existing string.
  - `template<typename T > std::string operator+ (const Framework::EnumMapWrapper< T > &lhs, const std::string &rhs)`  
Concatenate an existing string to the string representation of an enumeration.
  - `template<typename T > bool operator== (const std::string &lhs, const ConstEnumMapWrapper< T > &rhs)`  
Determine if a string and the string representation of an enumeration are equal.
  - `template<typename T > bool operator== (const ConstEnumMapWrapper< T > &lhs, const std::string &rhs)`  
Determine if a string representation of an enumeration and a string are equal.
  - `template<typename T > bool operator!= (const std::string &lhs, const ConstEnumMapWrapper< T > &rhs)`  
Determine if a string and the string representation of an enumeration are not equal.
  - `template<typename T > bool operator!= (const ConstEnumMapWrapper< T > &lhs, const std::string &rhs)`  
Determine if a string representation of an enumeration and a string are not equal.
  - `template<typename T > std::ostream & operator<< (std::ostream &stream, const Framework::ConstEnumMapWrapper< T > &kind)`  
Append the string representation of an enumeration into a stream.
  - `template<typename T > std::string operator+ (const std::string &lhs, const Framework::ConstEnumMapWrapper< T > &rhs)`  
Concatenate the string representation of an enumeration to an existing string.
  - `template<typename T > std::string operator+ (const Framework::ConstEnumMapWrapper< T > &lhs, const std::string &rhs)`  
Concatenate an existing string to the string representation of an enumeration.

## E.6.1 Detailed Description

Information about the framework.

## E.6.2 Function Documentation

**std::string BiometricEvaluation::Framework::getCompileDate ( )**

Date when this framework was compiled.

Returns

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

**std::string BiometricEvaluation::Framework::getCompiler ( )**

Compiler used to compile this framework.

Returns

The name of the compiler used to compile this framework.

**std::string BiometricEvaluation::Framework::getCompilerVersion ( )**

Version string of compiler used to compile this framework.

Returns

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

**std::string BiometricEvaluation::Framework::getCompileTime ( )**

[Time](#) when this framework was compiled.

Returns

[Time](#) when this framework was compiled, in the form "HH:MM:SS"

**unsigned int BiometricEvaluation::Framework::getMajorVersion ( )**

[Framework](#) major version.

Returns

The major version number of the BiometricFramework

**unsigned int BiometricEvaluation::Framework::getMinorVersion ( )**

[Framework](#) minor version.

Returns

The minor version of the [BiometricEvaluation](#) framework.

**template<typename T > bool BiometricEvaluation::Framework::operator!= ( const std::string & lhs, const EnumMapWrapper< T > & rhs )**

Determine if a string and the string representation of an enumeration are not equal.

Parameters

<i>lhs</i>	The string to compare to the enumeration.
<i>rhs</i>	The enumeration to compare to the string.

Returns

true if lhs is not equal to the string representation of rhs, false otherwise.

Note

String comparison is case-sensitive.

**template<typename T > bool BiometricEvaluation::Framework::operator!= ( const EnumMapWrapper< T > &lhs, const std::string &rhs )**

Determine if a string representation of an enumeration and a string are not equal.

Parameters

<i>lhs</i>	The enumeration to compare to the string.
<i>rhs</i>	The string to compare to the enumeration.

Returns

true if rhs is not equal to the string representation of rhs, false otherwise.

Note

String comparison is case-sensitive.

**template<typename T > bool BiometricEvaluation::Framework::operator!= ( const std::string &lhs, const ConstEnumMapWrapper< T > &rhs )**

Determine if a string and the string representation of an enumeration are not equal.

Parameters

<i>lhs</i>	The string to compare to the enumeration.
<i>rhs</i>	The enumeration to compare to the string.

Returns

true if lhs is not equal to the string representation of rhs, false otherwise.

Note

String comparison is case-sensitive.

**template<typename T > bool BiometricEvaluation::Framework::operator!= ( const ConstEnumMapWrapper< T > &lhs, const std::string &rhs )**

Determine if a string representation of an enumeration and a string are not equal.

Parameters

<i>lhs</i>	The enumeration to compare to the string.
<i>rhs</i>	The string to compare to the enumeration.

Returns

true if rhs is not equal to the string representation of rhs, false otherwise.

Note

String comparison is case-sensitive.

**template<typename T > std::string BiometricEvaluation::Framework::operator+ ( const std::string & lhs, const Framework::EnumMapWrapper< T > & rhs )**

Concatenate the string representation of an enumeration to an existing string.

Parameters

<i>lhs</i>	Existing string.
<i>rhs</i>	Enumeration whose string representation should be concatenated.

Returns

String made by appending string representation of rhs to lhs.

**template<typename T > std::string BiometricEvaluation::Framework::operator+ ( const Framework::EnumMapWrapper< T > & lhs, const std::string & rhs )**

Concatenate an existing string to the string representation of an enumeration.

Parameters

<i>lhs</i>	Enumeration whose string representation should be concatenated.
<i>rhs</i>	Existing string.

Returns

String made by appending lhs to the string representation of rhs.

**template<typename T > std::string BiometricEvaluation::Framework::operator+ ( const std::string & lhs, const Framework::ConstEnumMapWrapper< T > & rhs )**

Concatenate the string representation of an enumeration to an existing string.

Parameters

<i>lhs</i>	Existing string.
<i>rhs</i>	Enumeration whose string representation should be concatenated.

Returns

String made by appending string representation of rhs to lhs.

**template<typename T > std::string BiometricEvaluation::Framework::operator+ ( const Framework::ConstEnumMapWrapper< T > & lhs, const std::string & rhs )**

Concatenate an existing string to the string representation of an enumeration.

Parameters

<i>lhs</i>	Enumeration whose string representation should be concatenated.
<i>rhs</i>	Existing string.

Returns

String made by appending lhs to the string representation of rhs.

**template<typename T > std::ostream & BiometricEvaluation::Framework::operator<< ( std::ostream & *stream*, const EnumMapWrapper< T > & *kind* )**

Append the string representation of an enumeration into a stream.

Parameters

<i>stream</i>	The stream in which the string representation of kind should be appended.
<i>kind</i>	The enumeration whose string representation should be appended to stream.

Returns

Reference to stream.

**template<typename T > std::ostream & BiometricEvaluation::Framework::operator<< ( std::ostream & *stream*, const Framework::ConstEnumMapWrapper< T > & *kind* )**

Append the string representation of an enumeration into a stream.

Parameters

<i>stream</i>	The stream in which the string representation of kind should be appended.
<i>kind</i>	The enumeration whose string representation should be appended to stream.

Returns

Reference to stream.

**template<typename T > bool BiometricEvaluation::Framework::operator==( const std::string & *lhs*, const EnumMapWrapper< T > & *rhs* )**

Determine if a string and the string representation of an enumeration are equal.

Parameters

<i>lhs</i>	The string to compare to the enumeration.
<i>rhs</i>	The enumeration to compare to the string.

Returns

true if lhs is equal to the string representation of rhs, false otherwise.

Note

String comparison is case-sensitive.

**template<typename T > bool BiometricEvaluation::Framework::operator==( const EnumMapWrapper< T > & *lhs*, const std::string & *rhs* )**

Determine if a string representation of an enumeration and a string are equal.

Parameters

<i>lhs</i>	The enumeration to compare to the string.
<i>rhs</i>	The string to compare to the enumeration.

Returns

true if rhs is equal to the string representation of rhs, false otherwise.

Note

String comparison is case-sensitive.

**template<typename T > bool BiometricEvaluation::Framework::operator==( const std::string & *lhs*, const ConstEnumMapWrapper< T > & *rhs* )**

Determine if a string and the string representation of an enumeration are equal.

Parameters

<i>lhs</i>	The string to compare to the enumeration.
<i>rhs</i>	The enumeration to compare to the string.

Returns

true if lhs is equal to the string representation of rhs, false otherwise.

Note

String comparison is case-sensitive.

**template<typename T > bool BiometricEvaluation::Framework::operator==( const ConstEnumMapWrapper< T > & *lhs*, const std::string & *rhs* )**

Determine if a string representation of an enumeration and a string are equal.

Parameters

<i>lhs</i>	The enumeration to compare to the string.
<i>rhs</i>	The string to compare to the enumeration.

Returns

true if rhs is equal to the string representation of rhs, false otherwise.

Note

String comparison is case-sensitive.

## E.7 BiometricEvaluation::Image Namespace Reference

Basic information relating to images.

## Classes

- class [BMP](#)  
*A BMP-encoded image.*
- struct [Coordinate](#)  
*A structure to contain a two-dimensional coordinate without a specified origin.*
- 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.*
- struct [Resolution](#)  
*A structure to represent the resolution of an image.*
- struct [Size](#)  
*A structure to represent the size of an image, in pixels.*
- class [WSQ](#)  
*A WSQ-encoded image.*

## Typedefs

- using **Coordinate** = struct [Coordinate](#)
- using **CoordinateSet** = std::vector< [Image::Coordinate](#) >
- using **Size** = struct [Size](#)
- using **Resolution** = struct [Resolution](#)

## Enumerations

- enum [CompressionAlgorithm](#) {  
**None** = 0, **Facsimile** = 1, **WSQ20** = 2, **JPEGB** = 3,  
**JPEGL** = 4, **JP2** = 5, **JP2L** = 6, **PNG** = 7,  
**NetPBM** = 8, **BMP** = 9 }

## Functions

- 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](#) &)
- float [distance](#) (const [Coordinate](#) &p1, const [Coordinate](#) &p2)  
*Calculate the distance between two points.*



### E.7.1 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.

### E.7.2 Enumeration Type Documentation

**enum BiometricEvaluation::Image::CompressionAlgorithm** [**strong**]

[Image](#) compression algorithms.

### E.7.3 Function Documentation

**float BiometricEvaluation::Image::distance** ( **const** Coordinate & *p1*, **const** Coordinate & *p2* )

Calculate the distance between two points.

Parameters

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

Returns

Distance between p1 and p2.

**std::ostream& BiometricEvaluation::Image::operator<<** ( **std::ostream** & *stream*, **const** CoordinateSet & *coordinates* )

Output stream overload for CoordinateSet.

Parameters

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

Returns

stream with a coordinates textual representation appended.

## E.8 BiometricEvaluation::IO Namespace Reference

Input/Output functionality.

### Namespaces

- [Utility](#)

## Classes

- 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 [FileLogCabinet](#)
- class [FileLogsheet](#)  
*A class to represent a single logging mechanism with a file as the backing store.*
- class [FileRecordStore](#)
- class [GZip](#)  
*Compressor for gzip compression from zlib.*
- class [ListRecordStore](#)  
*[RecordStore](#) that reads a list of keys from a text file, and retrieves the data from another [RecordStore](#).*
- class [Logsheet](#)  
*A class to represent a logging mechanism.*
- struct [ManifestEntry](#)
- 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 [RecordStoreIterator](#)  
*Generic ForwardIterator for all [RecordStores](#).*
- class [SQLiteRecordStore](#)  
*A [RecordStore](#) implementation using a SQLite database as the underlying record storage system.*
- class [SysLogsheet](#)  
*A class to represent a single logging mechanism to a logging service on the network.*

## Typedefs

- using **ManifestEntry** = struct [ManifestEntry](#)
- using **ManifestMap** = [Memory::OrderedMap](#)< std::string, [ManifestEntry](#) >

### E.8.1 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.

## E.8.2 Typedef Documentation

**using BiometricEvaluation::IO::ManifestMap = typedef Memory::OrderedMap<std::string, ManifestEntry>**

Convenience alias for storing the manifest

## E.9 BiometricEvaluation::IO::Utility Namespace Reference

### Functions

- void [removeDirectory](#) (const std::string &directory, const std::string &prefix)  
*Remove a directory using directory name and parent pathname.*
- void [removeDirectory](#) (const std::string &pathname)  
*Remove a directory using a complete pathname.*
- void [copyDirectoryContents](#) (const std::string &sourcepath, const std::string &targetpath, const bool removeSource=false)  
*Copy the contents of a directory, optionally deleting the source directory contents when done.*
- void [setAsideName](#) (const std::string &name)  
*Set aside a file or directory name.*
- uint64\_t [getFileSize](#) (const std::string &pathname)
- uint64\_t [sumDirectoryUsage](#) (const std::string &pathname)
- bool [fileExists](#) (const std::string &pathname)
- bool [pathIsDirectory](#) (const std::string &pathname)
- int [makePath](#) (const std::string &path, const mode\_t mode)  
*Create an entire directory tree.*
- [Memory::uint8Array](#) [readFile](#) (const std::string &path, std::ios\_base::openmode mode=std::ios\_base::binary)  
*Read the contents of a file into a buffer.*
- void [writeFile](#) (const uint8\_t \*data, const size\_t size, const std::string &path, std::ios\_base::openmode mode=std::ios\_base::binary)  
*Write the contents of a buffer to a file.*
- void [writeFile](#) (const [Memory::uint8Array](#) data, const std::string &path, std::ios\_base::openmode mode=std::ios\_base::binary)  
*Write the contents of a buffer to a file.*
- bool [isReadable](#) (const std::string &pathname)  
*Determine if a file can be opened with read permission.*
- bool [isWritable](#) (const std::string &pathname)  
*Determine if a file can be opened with read/write permission.*
- std::string [createTemporaryFile](#) (const std::string &prefix="", const std::string &parentDir="/tmp")  
*Create a temporary file.*
- FILE \* [createTemporaryFile](#) (std::string &path, const std::string &prefix="", const std::string &parentDir="/tmp")  
*Create a temporary file.*

### E.9.1 Detailed Description

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

## E.9.2 Function Documentation

**void BiometricEvaluation::IO::Utility::copyDirectoryContents ( const std::string & *sourcepath*, const std::string & *targetpath*, const bool *removesource* = *false* )**

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

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

**std::string BiometricEvaluation::IO::Utility::createTemporaryFile ( const std::string & *prefix* = "", const std::string & *parentDir* = "/tmp" )**

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

<a href="#"><i>Error::FileError</i></a>	Could not create or close temporary file.
<a href="#"><i>Error::MemoryError</i></a>	<a href="#">Error</a> 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.

**FILE\* BiometricEvaluation::IO::Utility::createTemporaryFile ( std::string & *path*, const std::string & *prefix* = "", const std::string & *parentDir* = "/tmp" )**

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.

**bool BiometricEvaluation::IO::Utility::fileExists ( const std::string & *pathname* )**

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.
-----------------------------	--

**uint64\_t BiometricEvaluation::IO::Utility::getFileSize ( const std::string & *pathname* )**

Get the size of a file.

## Parameters

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

## Returns

The file size.

## Exceptions

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

**bool BiometricEvaluation::IO::Utility::isReadable ( const std::string & *pathname* )**

Determine if a file can be opened with read permission.

## Parameters

<code>in</code>	<code>pathname</code>	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\(\)](#)

### **bool BiometricEvaluation::IO::Utility::isWritable ( const std::string & *pathname* )**

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

## Parameters

<code>in</code>	<code>pathname</code>	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\(\)](#)

### **int BiometricEvaluation::IO::Utility::makePath ( const std::string & *path*, const mode\_t *mode* )**

Create an entire directory tree.

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

## Parameters

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

## Returns

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

### **Memory::uint8Array BiometricEvaluation::IO::Utility::readFile ( const std::string & *path*, std::ios\_base::openmode *mode* = *std::ios\_base::binary* )**

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.

**void BiometricEvaluation::IO::Utility::removeDirectory ( const std::string & *directory*, const std::string & *prefix* )**

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.

**void BiometricEvaluation::IO::Utility::removeDirectory ( const std::string & *pathname* )**

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.

**void BiometricEvaluation::IO::Utility::setAsideName ( const std::string & *name* )**

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

<code>in</code>	<code>name</code>	The path name of the file or directory to be set aside.
-----------------	-------------------	---

Exceptions

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

**uint64\_t BiometricEvaluation::IO::Utility::sumDirectoryUsage ( const std::string & *pathname* )**

Get the sum of the sizes of all files and directories in a given path.

Parameters

<code>in</code>	<code>pathname</code>	The name of the directory to be sized.
-----------------	-----------------------	--

Returns

The sum of file and directory sizes.

Exceptions

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

**void BiometricEvaluation::IO::Utility::writeFile ( const uint8\_t \* *data*, const size\_t *size*, const std::string & *path*, std::ios\_base::openmode *mode* = *std::ios\_base::binary* )**

Write the contents of a buffer to a file.

Parameters

<code>data</code>	Data buffer to write.
<code>size</code>	Size of data.
<code>path</code>	Path to file to create with contents of data.
<code>mode</code>	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.

**void BiometricEvaluation::IO::Utility::writeFile ( const Memory::uint8Array *data*, const std::string & *path*, std::ios\_base::openmode *mode* = *std::ios\_base::binary* )**

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.

## E.10 BiometricEvaluation::Iris Namespace Reference

Biometric information relating to iris images and derived information.

### Classes

- class [INCITSView](#)  
*A class to represent single iris view and derived information.*
- class [ISO2011View](#)  
*A class to represent single iris view and derived information.*

### Enumerations

- enum [CaptureDeviceTechnology](#) { **Unknown** = 0, **CMOSCCD** = 1 }  
*Capture device technology identifiers.*
- enum [EyeLabel](#) { **Undefined** = 0, **Right** = 1, **Left** = 2 }  
*Eye label.*
- enum [ImageType](#) { **Uncropped** = 1, **VGA** = 2, **Cropped** = 3, **CroppedMasked** = 7 }  
*Iris image type classification codes.*
- enum [Orientation](#) { **Undefined** = 0, **Base** = 1, **Flipped** = 2 }  
*Iris horizontal orientation classification codes.*
- enum [ImageCompression](#) { **Undefined** = 0, **LosslessNone** = 1, **Lossy** = 2 }  
*Iris image compression type.*
- enum [CameraRange](#) { **Unassigned** = 0, **Failed** = 1, **Overflow** = 2 }  
*Range from camera lens center to subject iris.*

### E.10.1 Detailed Description

Biometric information relating to iris images and derived information.

The [Iris](#) package gathers all iris related matters, including classes to represent iris information and helper functions for conversion between biometric representations. Contained within this namespace are classes to represent specific record formats, such as ISO 19794-6.

## E.11 BiometricEvaluation::Memory Namespace Reference

Support for memory-related operations.

## Namespaces

- [AutoArrayUtility](#)

## Classes

- class [AutoArray](#)  
*A C-style array wrapped in the facade of a C++ STL container.*
- class [AutoArrayIterator](#)  
*RandomAccessIterator for any [AutoArray](#).*
- class [AutoBuffer](#)
- class [IndexedBuffer](#)  
*Manage a memory buffer with an index.*
- class [OrderedMap](#)
- class [OrderedMapConstIterator](#)
- class [OrderedMapIterator](#)

## Typedefs

- using **uint8Array** = [AutoArray](#)< uint8\_t >
- using **uint16Array** = [AutoArray](#)< uint16\_t >
- using **uint32Array** = [AutoArray](#)< uint32\_t >

### E.11.1 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.

## E.12 BiometricEvaluation::Memory::AutoArrayUtility Namespace Reference

### Functions

- template<typename T, typename = typename std::enable\_if<std::is\_same<T, uint8\_t>::value || std::is\_same<T, char>::value>↵  
::type>  
char \* **cstr** (const [AutoArray](#)< T > &rahc)  
*Cast an [AutoArray](#) of uint8\_t or char to a char\*.*
- template<typename T, typename = typename std::enable\_if<std::is\_same<T, uint8\_t>::value || std::is\_same<T, char>::value>↵  
::type>  
std::string **getString** (const [AutoArray](#)< T > &aa, typename [AutoArray](#)< T >::size\_type count)  
*Convert a uint8\_t or char [AutoArray](#) to a string.*
- template<typename T, typename = typename std::enable\_if<std::is\_same<T, uint8\_t>::value || std::is\_same<T, char>::value>↵  
::type>  
void **setString** ([AutoArray](#)< T > &aa, const std::string &str)  
*Copy a string into an AutoArray of uint8\_t or char.*
- template<typename T, typename = typename std::enable\_if<std::is\_same<T, uint8\_t>::value || std::is\_same<T, char>::value>↵  
::type>  
void **setString** ([AutoArray](#)< T > &aa, const char \*str,...)  
*Copy a string into an AutoArray of uint8\_t or char.*

### E.12.1 Detailed Description

Convenience functions for AutoArrays.

### E.12.2 Function Documentation

```
template<typename T, typename = typename std::enable_if<std::is_same<T, uint8_t>::value ||
std::is_same<T, char>::value>::type> char* BiometricEvaluation::Memory::AutoArrayUtility::cstr (
const AutoArray< T > &rahc ) [inline]
```

Cast an [AutoArray](#) of uint8\_t or char to a char\*.

Parameters

<i>rahc</i>	<a href="#">AutoArray</a> to cast.
-------------	------------------------------------

Returns

rahc casted as a char\*.

```
template<typename T, typename = typename std::enable_if<std::is_same<T, uint8_t>::value ||
std::is_same<T, char>::value>::type> std::string BiometricEvaluation::Memory::AutoArrayUtility::getString (
const AutoArray< T > &aa, typename AutoArray< T >::size_type count )
[inline]
```

Convert a uint8\_t or char [AutoArray](#) to a string.

Parameters

<i>aa</i>	<a href="#">AutoArray</a> to stringify.
<i>count</i>	Last byte of aa to include in the returned string.

Returns

String representation of aa.

Exceptions

<a href="#">Error::MemoryError</a>	count > aa.size()
------------------------------------	-------------------

```
template<typename T, typename = typename std::enable_if<std::is_same<T, uint8_t>::value || std::is_same<T, char>::value>::type> void BiometricEvaluation::Memory::AutoArrayUtility::setString (
AutoArray< T > &aa, const std::string &str ) [inline]
```

Copy a string into an AutoArray of uint8\_t or char.

Parameters

<i>aa</i>	<a href="#">AutoArray</a> whose contents will be replaced with str.
<i>str</i>	String to assign to <a href="#">AutoArray</a> .

```
template<typename T, typename = typename std::enable_if<std::is_same<T, uint8_t>::value || std::is_same<T, char>::value>::type> void BiometricEvaluation::Memory::AutoArrayUtility::setString (
AutoArray< T > &aa, const char *str, ... ) [inline]
```

Copy a string into an AutoArray of uint8\_t or char.

## Parameters

<i>aa</i>	<a href="#">AutoArray</a> whose contents will be replaced with str.
<i>str</i>	printf-style format string.
<i>...</i>	Variable list of arguments for printf formatting.

## E.13 BiometricEvaluation::MPI Namespace Reference

Common declarations and functions for the MPI-based functionality.

### Classes

- class [Distributor](#)  
*A class to represent an [MPI](#) task that distributes work to other tasks.*
- class [MessageTag](#)  
*The types of messages sent between [MPI](#) task processes.*
- class [Receiver](#)  
*A class to represent an [MPI](#) task that receives [WorkPackages](#) containers from the [Distributor](#).*
- class [RecordProcessor](#)  
*An implementation of a work package processor that will extract record store keys, and optionally, values, from a [WorkPackage](#).*
- class [RecordStoreDistributor](#)  
*An implementation of the [Distributor](#) abstraction that uses a record store for input to create the work packages.*
- class [RecordStoreResources](#)  
*A class to represent a set of resources needed by an [MPI](#) program using a [RecordStore](#) for input.*
- class [Resources](#)
- class [Runtime](#)  
*[Runtime](#) support for the startup/shutdown of [MPI](#) jobs.*
- class [TaskCommand](#)  
*The command given to an [MPI](#) task.*
- class [TaskStatus](#)  
*The status of an [MPI](#) distributor or receiver task.*
- class [WorkPackage](#)  
*A class to represent a piece of work to be acted upon by a processor.*
- class [WorkPackageProcessor](#)  
*Represents an object that processes the contents of a work package.*

### Functions

- `std::string generateUniqueID ()`  
*Obtain a unique ID for the current process.*
- `void printStatus (const std::string &message)`  
*Print a status message to stdout.*
- `void logEntry (IO::Logsheet &logsheet)`  
*Send the current log stream to the log device as a debug entry.*
- `void logMessage (IO::Logsheet &logsheet, const std::string &message)`

*Send a log message to the given Logsheet as a debug entry.*

- `std::shared_ptr`  
< [BiometricEvaluation::IO::Logsheet](#) > `openLogsheet` (const std::string &url, const std::string &description)

*Open a Logsheet object for a component of the [MPI](#) framework.*

## Variables

- bool **Exit**
- bool **QuickExit**
- bool **TermExit**

### E.13.1 Detailed Description

Common declarations and functions for the MPI-based functionality.

### E.13.2 Function Documentation

**std::string BiometricEvaluation::MPI::generateUniqueID ( )**

Obtain a unique ID for the current process.

The ID is a string that is based on the host name, [MPI](#) rank, and process ID, formatted in a manner that can be used to uniquely name files.

Returns

The unique ID for the process.

**void BiometricEvaluation::MPI::logEntry ( IO::Logsheet & logsheet )**

Send the current log stream to the log device as a debug entry.

Log messages may be streamed into the Logsheet and written as debug messages to aid tracing. In order to prevent log errors interfering with the [MPI](#) job, errors are managed, and therefore, log messages may stop if the Logsheet has failed.

Parameters

<code>in</code>	<code>logsheet</code>	The open Logsheet to write into.
-----------------	-----------------------	----------------------------------

**void BiometricEvaluation::MPI::logMessage ( IO::Logsheet & logsheet, const std::string & message )**

Send a log message to the given Logsheet as a debug entry.

In order to prevent log errors interfering with the [MPI](#) job, errors are managed, and therefore, log messages may stop if the Logsheet has failed.

Parameters

<code>in</code>	<code>logsheet</code>	The open Logsheet to write into.
<code>in</code>	<code>message</code>	The log message.

**std::shared\_ptr<BiometricEvaluation::IO::Logsheet> BiometricEvaluation::MPI::openLogsheet ( const std::string & url, const std::string & description )**

Open a Logsheet object for a component of the [MPI](#) framework.

If the empty string is passed in as the URL, then a Null Logsheet object is returned.

## Parameters

in	<i>url</i>	The Uniform Resource Locator for the Logsheet.
in	<i>description</i>	The description of the Logsheet.

## Returns

Shared pointer to the Logsheet object.

## Exceptions

<a href="#"><i>Error::ParameterError</i></a>	Invalid URL.
<a href="#"><i>Error::Exception</i></a>	Failed to create the Logsheet object. The exception string will contain more information.

**void BiometricEvaluation::MPI::printStats ( const std::string & *message* )**

Print a status message to stdout.

## Parameters

in	<i>message</i>	The message to be printed.
----	----------------	----------------------------

## E.14 BiometricEvaluation::Process Namespace Reference

[Process](#) information and controls.

### Classes

- class [CommandCenter](#)
- class [CommandParser](#)
- 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 [MessageCenter](#)
- class [MessageCenterListener](#)
- class [MessageCenterReceiver](#)
  - Receives message from a client, forwarding to the central [MessageCenter](#).
- class [POSIXThreadManager](#)
  - [\*Manager\*](#) implementation that starts *Workers* in POSIX threads.
- class [POSIXThreadWorkerController](#)
  - Decorated [Worker](#) returned from a [Process::POSIXThreadManager](#).
- class [Semaphore](#)
  - Represent a semaphore that can be used for interprocess communication.
- 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

- using [ParameterList](#) = std::map< std::string, std::shared\_ptr< void >>

### E.14.1 Detailed Description

[Process](#) information and controls.

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

### E.14.2 Typedef Documentation

using [BiometricEvaluation::Process::ParameterList](#) = typedef std::map<std::string, std::shared\_ptr<void>>

Convenience alias for parameter lists to child routines

## E.15 BiometricEvaluation::System Namespace Reference

Operating system, hardware, etc.

## Functions

- uint32\_t [getCPUCount](#) ()  
*Obtain the number of central processing units that are online. Typically, this is the total CPU core count for the system.*
- uint64\_t [getRealMemorySize](#) ()  
*Obtain the amount of real memory in the system.*
- double [getLoadAverage](#) ()  
*Obtain the system load average for the last minute.*

### E.15.1 Detailed Description

Operating system, hardware, etc.

The [System](#) package gathers all system related matters, such as the operating system name, number of CPUs, etc.

### E.15.2 Function Documentation

uint32\_t [BiometricEvaluation::System::getCPUCount](#) ( )

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

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

### **double BiometricEvaluation::System::getLoadAverage ( )**

Obtain the system load average for the last minute.

Returns

The system load average.

Exceptions

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

### **uint64\_t BiometricEvaluation::System::getRealMemorySize ( )**

Obtain the amount of real memory in the system.

Returns

The real memory size, in kilobytes.

Exceptions

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

## **E.16 BiometricEvaluation::Text Namespace Reference**

[Text](#) processing for string objects.

### **Functions**

- void [removeLeadingTrailingWhitespace](#) (std::string &s)  
*Remove lead and trailing white space from a string object.*
- std::string [digest](#) (const std::string &s, const std::string &digest="md5")  
*Compute the digest of a string.*
- std::string [digest](#) (const void \*buffer, const size\_t buffer\_size, const std::string &digest="md5")  
*Compute the digest of a memory buffer.*
- std::vector< std::string > [split](#) (const std::string &str, const char delimiter, bool escape=true)  
*Return tokens bound by delimiters and the beginning and end of a string.*
- std::string [basename](#) (const std::string &path)  
*Extract the filename component of a pathname.*
- std::string [dirname](#) (const std::string &path)  
*Extract the directory component of a pathname.*
- bool [caseInsensitiveCompare](#) (const std::string &str1, const std::string &str2)  
*Compare two ASCII-encoded strings.*

### E.16.1 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.

### E.16.2 Function Documentation

**std::string BiometricEvaluation::Text::basename ( const std::string & *path* )**

Extract the filename component of a pathname.

Returns the component following the final '/'. Trailing '/' characters are not counted as part of the path-name.

Parameters

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

Returns

Filename portion of path.

**bool BiometricEvaluation::Text::caseInsensitiveCompare ( const std::string & *str1*, const std::string & *str2* )**

Compare two ASCII-encoded strings.

Parameters

<i>str1</i>	First string to compare.
<i>str2</i>	Second string to compare.

Returns

true if str1 and str2 are equal other than case, false otherwise.

**std::string BiometricEvaluation::Text::digest ( const std::string & *s*, const std::string & *digest* = "md5" )**

Compute the digest of a string.

Parameters

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

Exceptions

<a href="#">Error::MemoryError</a>	Could not allocate memory to store digest.
<a href="#">Error::NotImplemented</a>	The value of digest is not a supported digest.
<a href="#">Error::StrategyError</a>	An error occurred while obtaining the digest.

Returns

An ASCII representation of the hex digits composing the digest.

```
std::string BiometricEvaluation::Text::digest ( const void * buffer, const size_t buffer_size, const  
std::string & digest = "md5" )
```

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.

## Exceptions

<a href="#"><i>Error::MemoryError</i></a>	Could not allocate memory to store digest.
<a href="#"><i>Error::NotImplemented</i></a>	The value of digest is not a supported digest.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred while obtaining the digest.

## Returns

An ASCII representation of the hex digits composing the digest.

**std::string BiometricEvaluation::Text::dirname ( const std::string & *path* )**

Extract the directory component of a pathname.

Returns the string up to, but not including, the final '/'.

## Parameters

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

## Returns

Directory portion of path.

**std::vector<std::string> BiometricEvaluation::Text::split ( const std::string & *str*, const char *delimiter*, bool *escape* = *true* )**

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 will still contain one item, *str*.

## E.17 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

- `std::string getCurrentTime ()`
- `std::string getCurrentDate ()`
- `std::string getCurrentDateAndTime ()`
- `std::string getCurrentCalendarInformation (const std::string &formatString)`  
*Obtain customized calendar information.*
- `std::string put\_time (const struct tm *tmb, const char *fmt)`  
*Manual implementation of `std::put_time`.*
- `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 NanosecondsPerMicrosecond = 1000`
- `const int MicrosecondsPerSecond = 1000000`
- `const int MicrosecondsPerMillisecond = 1000`
- `const int MillisecondsPerSecond = 1000`

### E.17.1 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.

### E.17.2 Function Documentation

**`std::string BiometricEvaluation::Time::getCurrentCalendarInformation ( const std::string &formatString )`**

Obtain customized calendar information.

Parameters

<i>formatString</i>	A C++11 <code>put_time</code> -compatible format string.
---------------------	--

Returns

The current calendar information formatted as specified in `formatString`.

Note

Return value is undefined if format string is invalid.

**std::string BiometricEvaluation::Time::getCurrentDate ( )**

Returns

The current ISO 8601 date as a string.

**std::string BiometricEvaluation::Time::getCurrentDateAndTime ( )**

Returns

The standard locale current date and time as a string.

**std::string BiometricEvaluation::Time::getCurrentTime ( )**

Returns

The current ISO 8601 time as a string.

**std::string BiometricEvaluation::Time::put\_time ( const struct tm \* *tmb*, const char \* *fmt* )**

Manual implementation of std::put\_time.

Note

Exists because g++ does not currently implement put\_time ([http://gcc.gnu.org/bugzilla/show\\_bug.cgi?id=54354](http://gcc.gnu.org/bugzilla/show_bug.cgi?id=54354))

## E.18 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)  
*Output stream overload for DeviceMonitoringMode.*
- std::ostream & [operator<<](#) (std::ostream &stream, const [AN2KViewVariableResolution::AN2KQualityMetric](#) &qm)  
*Output stream overload for AN2KQualityMetric.*

### E.18.1 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.

### E.18.2 Function Documentation

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

Output stream overload for AN2KQualityMetric.

## Parameters

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

## Returns

stream with a qm textual representation appended.





## Appendix F

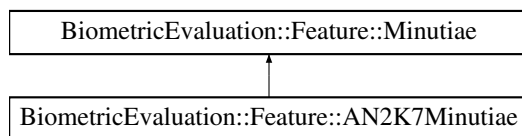
# Class Documentation

### F.1 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

- struct [FingerprintReadingSystem](#)  
*Representation of information about a fingerprint reader system.*
- class [PatternClassification](#)  
*Pattern classification codes.*

#### Public Types

- enum [EncodingMethod](#) { [EncodingMethod::Automatic](#) = 0, [EncodingMethod::AutomaticUnedited](#), [EncodingMethod::AutomaticEdited](#), [Manual](#) }
- using [PatternClassificationSet](#) = std::vector< [PatternClassification::Entry](#) >
- using [FingerprintReadingSystem](#) = struct [FingerprintReadingSystem](#)

#### Public Member Functions

- [AN2K7Minutiae](#) (const std::string &filename, int recordNumber)  
*Construct an AN2K7 [Minutiae](#) object from file data.*
- [AN2K7Minutiae](#) ([Memory::uint8Array](#) &buf, int recordNumber)  
*Construct an AN2K7 [Minutiae](#) object from data contained in a memory buffer.*
- [PatternClassificationSet](#) [getPatternClassificationSet](#) () const

- Obtain the set fingerprint pattern classifications.
- [FingerprintReadingSystem](#) [getOriginatingFingerprintReadingSystem](#) () const
- [MinutiaeFormat](#) [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](#) [convertPatternClassification](#) (const char \*fpc)  
Convert string read from AN2K record into a [PatternClassification](#).
- static  
[Finger::PatternClassification](#) [convertPatternClassification](#) (const [PatternClassification::Entry](#) &entry)  
Convert a standard [PatternClassification::Entry](#) to a [PatternClassification::Kind](#).
- static [EncodingMethod](#) [convertEncodingMethod](#) (const char \*mem)  
Convert string read from AN2K record into a [EncodingMethod](#).
- static [Image::Coordinate](#) [convertCoordinate](#) (const char \*str, bool calculateDistance=true)  
Obtain a [Coordinate](#) given an AN2K entry.

### F.1.1 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.

### F.1.2 Member Enumeration Documentation

**enum [BiometricEvaluation::Feature::AN2K7Minutiae::EncodingMethod](#) [strong]**

Methods for encoding minutiae data in an AN2K record.

Enumerator

- Automatic* No possible human interaction
- AutomaticUnedited* Editing possible, but not performed
- AutomaticEdited* Editing possible and was performed

### F.1.3 Constructor & Destructor Documentation

**[BiometricEvaluation::Feature::AN2K7Minutiae::AN2K7Minutiae](#) ( const std::string &filename, int recordNumber )**

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

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

**BiometricEvaluation::Feature::AN2K7Minutiae::AN2K7Minutiae ( Memory::uint8Array & *buf*, int *recordNumber* )**

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

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

## F.1.4 Member Function Documentation

**static Image::Coordinate BiometricEvaluation::Feature::AN2K7Minutiae::convertCoordinate ( const char \* *str*, bool *calculateDistance* = *true* ) [static]**

Obtain a Coordinate given an AN2K entry.

This AN2K entry is formatted as "XXXXYYYYY".

## Parameters

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

## Returns

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

## Exceptions

<a href="#"><i>Error::DataError</i></a>	Invalid format of <i>str</i> .
---	--------------------------------

**static EncodingMethod BiometricEvaluation::Feature::AN2K7Minutiae::convertEncodingMethod ( const char \* *mem* ) [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

<a href="#"><i>Error::DataError</i></a>	Invalid value for mem.
---	------------------------

**static Finger::PatternClassification BiometricEvaluation::Feature::AN2K7Minutiae::convertPattern↵  
Classification ( const char \* *fpc* ) [static]**

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

Parameters

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

Exceptions

<a href="#"><i>Error::DataError</i></a>	Invalid value for fpc.
---	------------------------

**static Finger::PatternClassification BiometricEvaluation::Feature::AN2K7Minutiae↵  
::convertPatternClassification ( const PatternClassification::Entry & *entry* )  
[static]**

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

Parameters

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

Exceptions

<a href="#"><i>Error::DataError</i></a>	Non-standard pattern classification entry.
---	--

**FingerprintReadingSystem BiometricEvaluation::Feature::AN2K7Minutiae::getOriginating↵  
FingerprintReadingSystem ( ) const**

Obtain the originating fingerprint reading system.

Exceptions

<a href="#"><i>Error::ObjectDoesNot↵ Exist</i></a>	The optional OFR field has been excluded.
--	---

**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 isPatternClassification↵  
Standard() to check.

## F.2 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 std::string &filename, int recordNumber)  
*Construct an [AN2KMinutiaeDataRecord](#) object from data contained in a file on disk.*
- [AN2KMinutiaeDataRecord](#) ([Memory::uint8Array](#) &buf, int recordNumber)  
*Construct an [AN2KMinutiaeDataRecord](#) object from data contained in a memory buffer.*
- std::shared\_ptr  
< [Feature::AN2K7Minutiae](#) > [getAN2K7Minutiae](#) () const  
*Obtain the "standard" minutiae data from this Type-9 Record (fields 9.005 - 9.012).*
- [ImpressionType](#) [getImpressionType](#) () const  
*Return impression type field from Type-9 Record.*
- std::map< uint16\_t,  
[Memory::uint8Array](#) > [getRegisteredVendorBlock](#) ([Feature::MinutiaeFormat](#) vendor) const  
*Obtain data recorded in a registered vendor minutiae block found in this Type-9 Record.*

### F.2.1 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).

### F.2.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::AN2KMinutiaeDataRecord** ( const std::string &filename, int recordNumber )

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

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

**BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::AN2KMinutiaeDataRecord** ( [Memory::uint8Array](#) &buf, int recordNumber )

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

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

## F.2.3 Member Function Documentation

**std::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.

**Impression 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.

**std::map<uint16\_t, Memory::uint8Array> BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getRegisteredVendorBlock ( Feature::MinutiaeFormat vendor ) const**

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

## Parameters

in	<i>vendor</i>	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

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

## F.3 BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric Struct Reference

A structure to represent an AN2K quality metric.

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

## Public Attributes

- [Finger::Position](#) **position**
- `uint8_t` **score**
- `uint16_t` **vendorID**
- `uint16_t` **productCode**

### F.3.1 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.

## F.4 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

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

## Public Member Functions

- [AN2KRecord](#) (const std::string filename)  
*Constructor taking an AN2K record from a file.*
- [AN2KRecord](#) ([Memory::uint8Array](#) &buf)  
*Constructor taking an AN2K record from a buffer.*
- std::string [getVersionNumber](#) () const
- std::string [getDate](#) () const
- std::string [getDestinationAgency](#) () const
- std::string [getOriginatingAgency](#) () const
- std::string [getTransactionControlNumber](#) () const
- std::string [getNativeScanningResolution](#) () const
- std::string [getNominalTransmittingResolution](#) () const
- `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 std::set< int > recordLocations (Memory::uint8Array &buf, const View::AN2KView::RecordType recordType)`  
*Find the position within a buffer of all Records of a particular type.*
- `static std::set< int > recordLocations (const ANSI_NIST *an2k, const View::AN2KView::RecordType recordType)`  
*Find the position within an ANSI\_NIST struct of all Records of a particular type.*

### F.4.1 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.

### F.4.2 Member Typedef Documentation

**using BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet = struct CharacterSet**

Convenience alias for struct [CharacterSet](#)

**using BiometricEvaluation::DataInterchange::AN2KRecord::DomainName = struct DomainName**

Convenience alias for struct [DomainName](#)

### F.4.3 Constructor & Destructor Documentation

**BiometricEvaluation::DataInterchange::AN2KRecord::AN2KRecord ( const std::string filename )**

Constructor taking an AN2K record from a file.

## Parameters

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

## Exceptions

<a href="#"><code>Error::FileError</code></a>	An error occurred when opening or reading the file.
<a href="#"><code>Error::DataError</code></a>	An error occurred when processing the AN2K record.

**BiometricEvaluation::DataInterchange::AN2KRecord::AN2KRecord ( Memory::uint8Array & buf )**

Constructor taking an AN2K record from a buffer.

## Parameters

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

## Exceptions

<a href="#"><code>Error::DataError</code></a>	An error occurred when processing the AN2K record.
---	--

**F.4.4 Member Function Documentation****std::string BiometricEvaluation::DataInterchange::AN2KRecord::getDate ( ) const**

## Returns

The date field in the Type-1 record.

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::getDestinationAgency ( ) const**

## Returns

The destination agency ID.

**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.

**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).

**uint32\_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerCaptureCount ( ) const**

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

Returns

The number of captures in the AN2K record.

**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.

**uint32\_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerLatentCount ( ) const**

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

Returns

The number of latents in the AN2K record.

**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.

**struct tm BiometricEvaluation::DataInterchange::AN2KRecord::getGreenwichMeanTime ( ) const**

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

Returns

struct tm encoding of the GMT field.

**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.

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::getNativeScanningResolution ( ) const**

Returns

The native scanning resolution.

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::getNominalTransmittingResolution ( ) const**

Returns

The nominal transmitting resolution.

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::getOriginatingAgency ( ) const**

Returns

The originating agency ID.

**uint8\_t BiometricEvaluation::DataInterchange::AN2KRecord::getPriority ( ) const**

Obtain the urgency with which a response is required.

Returns

Priority (1:High - 9:Low)

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::getTransactionControlNumber ( ) const**

Returns

The transaction control number.

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::getVersionNumber ( ) const**

Returns

The record version field in the Type-1 record.

**static std::set<int> BiometricEvaluation::DataInterchange::AN2KRecord::recordLocations ( Memory::uint8Array & *buf*, const View::AN2KView::RecordType *recordType* ) [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

<a href="#"><i>Error::DataError</i></a>	An error occurred when processing the AN2K record.
---	--

**static std::set<int> BiometricEvaluation::DataInterchange::AN2KRecord::recordLocations ( const ANSI\_NIST \* *an2k*, const View::AN2KView::RecordType *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

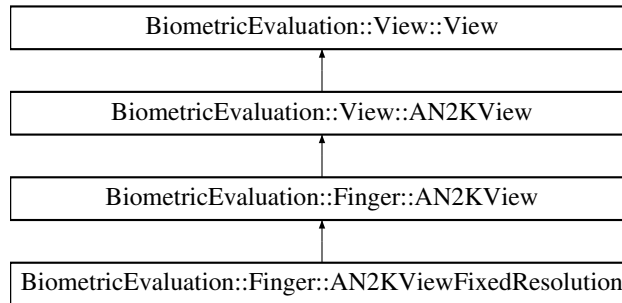
Set of integer positions within the ANSI\_NIST struct where a recordType Record is located.

## F.5 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

- std::vector  
< [\*AN2KMinutiaeDataRecord\*](#) > [\*getMinutiaeDataRecordSet\*](#) () const  
*Obtain the set of minutiae records.*
- Finger::PositionSet [\*getPositions\*](#) () const  
*Obtain the set of finger positions.*
- [\*Finger::Impression\*](#) [\*getImpressionType\*](#) () const  
*Obtain the finger impression code.*

### Static Public Member Functions

- static [\*Finger::Position\*](#) [\*convertPosition\*](#) (int *an2kFGP*)  
*Convert a compression algorithm indicator from an AN2K finger image record.*
- static Finger::PositionSet [\*populateFGP\*](#) (FIELD \**field*)  
*Read the finger positions from an AN2K record.*

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

## Protected Member Functions

- [AN2KView](#) (const std::string filename, const [RecordType](#) typeID, const uint32\_t recordNumber)  
*Construct an AN2K finger view from a file.*
- [AN2KView](#) ([Memory::uint8Array](#) &buf, const [RecordType](#) typeID, const uint32\_t recordNumber)  
*Construct an AN2K finger view from a buffer.*
- void [addMinutiaeDataRecord](#) ([Finger::AN2KMinutiaeDataRecord](#) &mdr)  
*Add a minutiae data record to the [AN2KMinutiaeDataRecord](#) set.*
- void [setPosition](#) ([Finger::PositionSet](#) &ps)  
*Add a position set to the collection of position sets.*
- void [setImpressionType](#) ([Finger::Impression](#) &imp)  
*Mutator for the impression type.*

## Additional Inherited Members

### F.5.1 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.

### F.5.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::AN2KView::AN2KView** ( const std::string *filename*, const [RecordType](#) *typeID*, const uint32\_t *recordNumber* ) **[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.

Parameters

in	<i>filename</i>	The name of the file containing the AN2K record.
in	<i>typeID</i>	The type of AN2K finger view: Type-3/Type-4/etc.
in	<i>recordNumber</i>	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

Exceptions

<a href="#">Error::ParameterError</a>	An invalid parameter was passed in.
---------------------------------------	-------------------------------------

<a href="#"><i>Error::DataError</i></a>	An error occurred when parsing the AN2K record.
<a href="#"><i>Error::FileError</i></a>	An error occurred when reading the file.

**BiometricEvaluation::Finger::AN2KView::AN2KView ( [Memory::uint8Array](#) & *buf*, const [RecordType](#) *typeID*, const [uint32\\_t](#) *recordNumber* ) [protected]**

Construct an AN2K finger view from a buffer.

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

Parameters

in	<i>buf</i>	The buffer containing the AN2K record.
in	<i>typeID</i>	The type of AN2K finger view: Type-3/Type-4/etc.
in	<i>recordNumber</i>	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

Exceptions

<a href="#"><i>Error::ParameterError</i></a>	An invalid parameter was passed in.
<a href="#"><i>Error::DataError</i></a>	An error occurred when parsing the AN2K record.

### F.5.3 Member Function Documentation

**void BiometricEvaluation::Finger::AN2KView::addMinutiaeDataRecord ( [Finger::AN2KMinutiaeDataRecord](#) & *mdr* ) [protected]**

Add a minutiae data record to the [AN2KMinutiaeDataRecord](#) set.

Parameters

in	<i>mdr</i>	The minutiae data record to be added.
----	------------	---------------------------------------

**static [Finger::FingerImageCode](#) BiometricEvaluation::Finger::AN2KView::convertFingerImageCode ( const char \* *str* ) [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

<a href="#"><i>Error::DataError</i></a>	The string contains an invalid image code.
---	--

**static [Finger::Position](#) BiometricEvaluation::Finger::AN2KView::convertPosition ( int *an2kFGP* ) [static]**

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

## Parameters

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

## Exceptions

<a href="#"><i>Error::DataError</i></a>	The position code is invalid.
---	-------------------------------

**Finger::Impression BiometricEvaluation::Finger::AN2KView::getImpressionType ( ) const**

Obtain the finger impression code.

## Returns

The finger impression code.

**std::vector<AN2KMinutiaeDataRecord> BiometricEvaluation::Finger::AN2KView::getMinutiae↵  
DataRecordSet ( ) const**

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.

## Returns

The vector of minutiae data records.

**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.

**static Finger::PositionSet BiometricEvaluation::Finger::AN2KView::populateFGP ( FIELD \*field )  
[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

<a href="#"><i>Error::DataError</i></a>	The data contains an invalid value.
---	-------------------------------------

**void BiometricEvaluation::Finger::AN2KView::setImpressionType ( Finger::Impression &imp )  
[protected]**

Mutator for the impression type.



Parameters

<code>in</code>	<code>imp</code>	The impression type for this finger view.
-----------------	------------------	---

**void BiometricEvaluation::Finger::AN2KView::setPositions ( Finger::PositionSet & *ps* )**  
**[protected]**

Add a position set to the collection of position sets.

Parameters

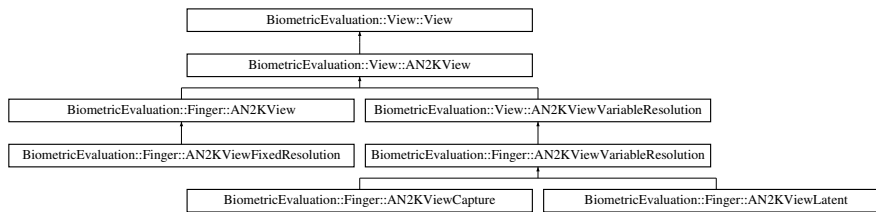
<code>in</code>	<code>ps</code>	The position set to be added.
-----------------	-----------------	-------------------------------

## F.6 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:



### Public Types

- enum [RecordType](#) : uint16\_t {  
**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 }
- enum [DeviceMonitoringMode](#) {  
**DeviceMonitoringMode::Controlled**, **DeviceMonitoringMode::Assisted**, **DeviceMonitoringMode::Observed**,  
**DeviceMonitoringMode::Unattended**,  
**DeviceMonitoringMode::Unknown**, **DeviceMonitoringMode::NA** }

*The level of human monitoring for the image capture device.*

### Public Member Functions

- [AN2KView](#) (const std::string filename, const [RecordType](#) typeID, const uint32\_t recordNumber)  
*Construct an AN2K view from a file.*
- [AN2KView](#) (Memory::uint8Array &buf, const [RecordType](#) typeID, const uint32\_t recordNumber)  
*Construct an AN2K view from a buffer.*
- std::vector  
< [Finger::AN2KMinutiaeDataRecord](#) > [getMinutiaeDataRecordSet](#) () const  
*Obtain the set of minutiae records.*

- [RecordType](#) [getRecordType](#) () const  
*Obtain the ANSI-NIST record type.*

## Static Public Member Functions

- static [DeviceMonitoringMode](#) [convertDeviceMonitoringMode](#) (const char \*dmm)  
*Convert a device monitoring mode indicator from an AN2K record.*
- static [Image::CompressionAlgorithm](#) [convertCompressionAlgorithm](#) (const uint16\_t recordType, const unsigned char \*an2kValue)  
*Convert a compression algorithm indicator from an AN2K finger image record.*

## 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.*

### F.6.1 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.

### F.6.2 Member Enumeration Documentation

**enum BiometricEvaluation::View::AN2KView::DeviceMonitoringMode [strong]**

The level of human monitoring for the image capture device.

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

**enum BiometricEvaluation::View::AN2KView::RecordType : uint16\_t [strong]**

The type of AN2K record.

### F.6.3 Constructor & Destructor Documentation

**BiometricEvaluation::View::AN2KView::AN2KView ( const std::string *filename*, const RecordType *typeID*, const uint32\_t *recordNumber* )**

Construct an AN2K view from a file.

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

**BiometricEvaluation::View::AN2KView::AN2KView ( Memory::uint8Array & *buf*, const RecordType *typeID*, const uint32\_t *recordNumber* )**

Construct an AN2K view from a buffer.

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

### F.6.4 Member Function Documentation

**static Image::CompressionAlgorithm BiometricEvaluation::View::AN2KView::convert↵  
CompressionAlgorithm ( const uint16\_t *recordType*, const unsigned char \* *an2kValue* )  
[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

<a href="#"><i>Error::DataError</i></a>	Invalid compression algorithm for record type.
<a href="#"><i>Error::ParameterError</i></a>	Invalid record type.

**static DeviceMonitoringMode BiometricEvaluation::View::AN2KView::convertDeviceMonitoringMode  
( const char \* *dmm* ) [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

<a href="#">Error::DataError</a>	Invalid format of dmm.
----------------------------------	------------------------

**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.

**std::vector<Finger::AN2KMinutiaeDataRecord> BiometricEvaluation::View::AN2KView::get↵  
MinutiaeDataRecordSet ( ) const**

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.

**RecordType BiometricEvaluation::View::AN2KView::getRecordType ( ) const**

Obtain the ANSI-NIST record type.

Returns

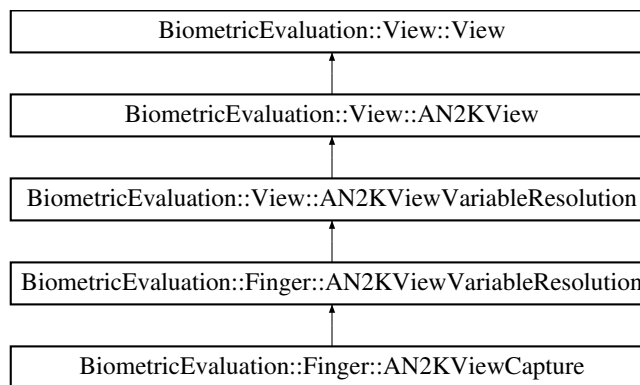
The type of record used to construct this object.

## F.7 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

- struct [FingerSegmentPosition](#)

*Locations of an individual finger segment in a slap.*

## Public Types

- enum [AmputatedBandaged](#) { [AmputatedBandaged::Amputated](#), [AmputatedBandaged::Bandaged](#), [AmputatedBandaged::NA](#) }
- using [FingerSegmentPosition](#) = struct [FingerSegmentPosition](#)
- using [FingerSegmentPositionSet](#) = std::vector< [FingerSegmentPosition](#) >

*Enumeration of the finger amputated or bandaged code, a reason that a capture could not be made.*

## Public Member Functions

- [AN2KViewCapture](#) (const std::string &filename, const uint32\_t recordNumber)  
*Construct an AN2K finger view from a file.*
- [AN2KViewCapture](#) ([Memory::uint8Array](#) &buf, const uint32\_t recordNumber)  
*Construct an AN2K finger view using from a memory buffer.*
- [QualityMetricSet](#) [extractNISTQuality](#) (const FIELD \*field)  
*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](#) [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](#) [convertAmputatedBandaged](#) (const char \*ampcd)  
*Convert string read from AN2K record into a AmputatedBandaged code.*
- static [FingerSegmentPosition](#) [convertFingerSegmentPosition](#) (const SUBFIELD \*sf)  
*Convert SUBFIELD read from AN2K record into a [FingerSegmentPosition](#) struct.*
- static [FingerSegmentPosition](#) [convertAlternateFingerSegmentPosition](#) (const SUBFIELD \*sf)  
*Convert SUBFIELD read from AN2K record into an AlternateFingerSegmentPosition struct.*

## Additional Inherited Members

### F.7.1 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).

## F.7.2 Member Enumeration Documentation

**enum BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged** [**strong**]

Enumeration of the finger amputated or bandaged code, a reason that a capture could not be made.

Enumerator

*Amputated* Amputation

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

*NA* Optional field – not specified

## F.7.3 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::AN2KViewCapture::AN2KViewCapture** ( **const** std::string & *filename*, **const** uint32\_t *recordNumber* )

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.

**BiometricEvaluation::Finger::AN2KViewCapture::AN2KViewCapture** ( **Memory::uint8Array** & *buf*, **const** uint32\_t *recordNumber* )

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.

## F.7.4 Member Function Documentation

**static FingerSegmentPosition BiometricEvaluation::Finger::AN2KViewCapture::convertAlternate↵  
FingerSegmentPosition** ( **const** SUBFIELD \* *sf* ) [**static**]

Convert SUBFIELD read from AN2K record into an AlternateFingerSegmentPosition struct.

Parameters

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

Exceptions

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

```
static AmputatedBandaged BiometricEvaluation::Finger::AN2KViewCapture::convertAmputated↵  
Bandaged ( const char * ampcd ) [static]
```

Convert string read from AN2K record into a AmputatedBandaged code.

## Parameters

in	ampcd	Value for amputated bandaged code read from an AN2K record.
----	-------	---

## Exceptions

<a href="#">Error::DataError</a>	Invalid value for ampcd.
----------------------------------	--------------------------

**static FingerSegmentPosition BiometricEvaluation::Finger::AN2KViewCapture::convertFinger↵  
SegmentPosition ( const SUBFIELD \* sf ) [static]**

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

## Parameters

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

## Exceptions

<a href="#">Error::DataError</a>	Invalid value within sf.
----------------------------------	--------------------------

**QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::extractNISTQuality ( const  
FIELD \* field )**

Extract the NQM information from an AN2K FIELD.

## Parameters

field	FIELD containing properly formatted NQM data
-------	--

## Returns

QualityMetricSet representation of field.

## Exceptions

<a href="#">Error::DataError</a>	Invalid format of field for NQM.
----------------------------------	----------------------------------

**FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getAlternateFinger↵  
SegmentPositionSet ( ) const**

## Returns

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

**AmputatedBandaged BiometricEvaluation::Finger::AN2KViewCapture::getAmputatedBandaged ( )  
const**

## Returns

Optional amputated or bandaged code.



**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

**FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getFingerSegmentPositionSet ( ) const**

Returns

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

**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.

**QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::getSegmentationQualityMetric ( ) const**

Obtain the segmentation quality metric for all segmented finger images.

Returns

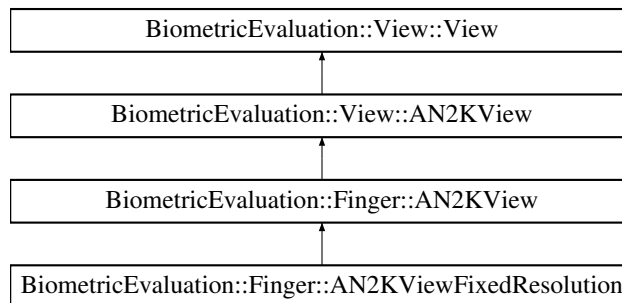
QualityMetricSet containing the segmentation quality metric for all segmented finger images.

## F.8 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](#) typeID, const uint32\_t record←  
Number)  
*Construct an AN2K finger view from a file.*
- [AN2KViewFixedResolution](#) ([Memory::uint8Array](#) &buf, const [RecordType](#) typeID, const uint32\_t record←  
Number)  
*Construct an AN2K finger view from a buffer.*

## Additional Inherited Members

### F.8.1 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.

### F.8.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution ( const std::string filename, const [RecordType](#) typeID, const uint32\_t recordNumber )**

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.

Parameters

in	<i>filename</i>	The name of the file containing the AN2K record.
in	<i>typeID</i>	The type of AN2K finger view: Type-3/Type-4/etc.
in	<i>recordNumber</i>	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

Exceptions

<a href="#">Error::ParameterError</a>	An invalid parameter was passed in.
<a href="#">Error::DataError</a>	An error occurred when parsing the AN2K record.
<a href="#">Error::FileError</a>	An error occurred when reading the file.

**BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution ( [Memory::uint8Array](#) &buf, const [RecordType](#) typeID, const uint32\_t recordNumber )**

Construct an AN2K finger view from a buffer.

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

Parameters

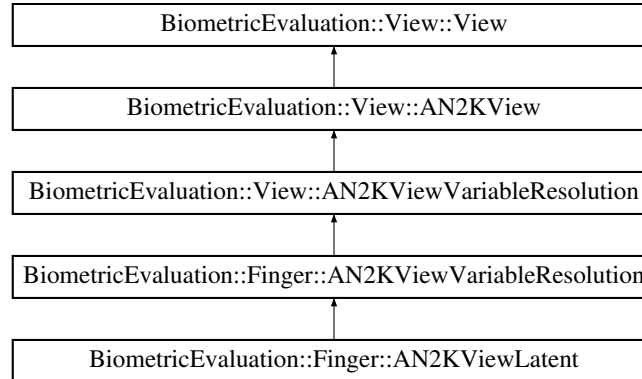
in	<i>buf</i>	The buffer containing the AN2K record.
in	<i>typeID</i>	The type of AN2K finger view: Type-3/Type-4/etc.
in	<i>recordNumber</i>	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

Exceptions

<a href="#"><i>Error::ParameterError</i></a>	An invalid parameter was passed in.
<a href="#"><i>Error::DataError</i></a>	An error occurred when parsing the AN2K record.

## F.9 BiometricEvaluation::Finger::AN2KViewLatent Class Reference

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewLatent:



### Public Member Functions

- [AN2KViewLatent](#) (const std::string &filename, const uint32\_t recordNumber)  
*Construct an AN2K finger view from a file.*
- [AN2KViewLatent](#) (Memory::uint8Array &buf, const uint32\_t recordNumber)  
*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.*

### Additional Inherited Members

#### F.9.1 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::AN2KViewLatent::AN2KViewLatent ( const std::string &filename, const uint32\_t recordNumber )**

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.

**BiometricEvaluation::Finger::AN2KViewLatent::AN2KViewLatent ( Memory::uint8Array &buf, const uint32\_t recordNumber )**

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.

## F.9.2 Member Function Documentation

**QualityMetricSet BiometricEvaluation::Finger::AN2KViewLatent::getLatentQualityMetric ( ) const**

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

Returns

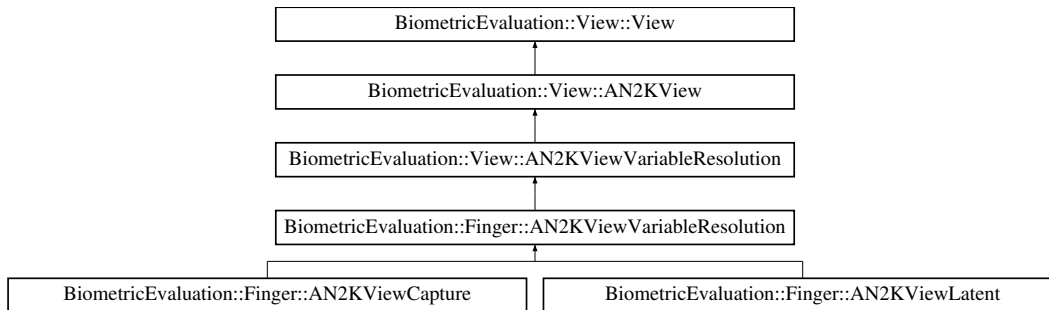
Latent quality metrics

## F.10 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

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

## Public Member Functions

- Finger::PositionSet [getPositions](#) ( ) const  
*Obtain the set of finger positions.*
- [Finger::Impression](#) [getImpressionType](#) ( ) const
- PrintPositionCoordinateSet [getPrintPositionCoordinates](#) ( ) const  
*Obtain print position coordinates.*

## Protected Member Functions

- [AN2KViewVariableResolution](#) (const std::string &filename, const [RecordType](#) typeID, const uint32\_t recordNumber)  
*Construct an AN2K finger view from a file.*
- [AN2KViewVariableResolution](#) ([Memory::uint8Array](#) &buf, const [RecordType](#) typeID, const uint32\_t recordNumber)  
*Construct an AN2K finger view from a buffer.*
- PositionDescriptors [getPositionDescriptors](#) () const

## Static Protected Member Functions

- static [PrintPositionCoordinate](#) [convertPrintPositionCoordinate](#) (SUBFIELD \*subfield)  
*Convert a print position coordinate AN2K subfield to a [PrintPositionCoordinate](#) object.*
- static PositionDescriptors [parsePositionDescriptors](#) (const [RecordType](#) typeID, const RECORD \*record)  
*Parse position descriptors from a record.*

## Additional Inherited Members

### F.10.1 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.

### F.10.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::AN2KViewVariableResolution::AN2KViewVariableResolution ( const std::string &filename, const [RecordType](#) typeID, const uint32\_t recordNumber ) [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.

Parameters

in	<i>filename</i>	The name of the file containing the AN2K record.
in	<i>typeID</i>	The type of AN2K finger view: Type-3/Type-4/etc.
in	<i>recordNumber</i>	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

Exceptions

<a href="#">Error::ParameterError</a>	An invalid parameter was passed in.
<a href="#">Error::DataError</a>	An error occurred when parsing the AN2K record.
<a href="#">Error::FileError</a>	An error occurred when reading the file.

**BiometricEvaluation::Finger::AN2KViewVariableResolution::AN2KViewVariableResolution ( [Memory::uint8Array](#) & buf, const [RecordType](#) typeID, const uint32\_t recordNumber ) [protected]**

Construct an AN2K finger view from a buffer.

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

## Parameters

in	<i>buf</i>	The buffer containing the AN2K record.
in	<i>typeID</i>	The type of AN2K finger view: Type-3/Type-4/etc.
in	<i>recordNumber</i>	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

## Exceptions

<a href="#"><i>Error::ParameterError</i></a>	An invalid parameter was passed in.
<a href="#"><i>Error::DataError</i></a>	An error occurred when parsing the AN2K record.

## F.10.3 Member Function Documentation

**static PrintPositionCoordinate BiometricEvaluation::Finger::AN2KViewVariableResolution::convertPrintPositionCoordinate ( SUBFIELD \* *subfield* ) [static], [protected]**

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

## Parameters

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

## Returns

Object representation of field.

## Exceptions

<a href="#"><i>Error::DataError</i></a>	Invalid data for a print position coordinate AN2K field.
---	--

**Finger::Impression BiometricEvaluation::Finger::AN2KViewVariableResolution::getImpressionType ( ) const**

## Returns

The finger impression code.

**PositionDescriptors BiometricEvaluation::Finger::AN2KViewVariableResolution::getPositionDescriptors ( ) const [protected]**

## Returns

The set of position descriptors.

**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.

**PrintPositionCoordinateSet** **BiometricEvaluation::Finger::AN2KViewVariableResolution::getPrint**↵  
**PositionCoordinates** ( ) **const**

Obtain print position coordinates.

Returns

Set of all PrintPositionCoordinates

**static PositionDescriptors** **BiometricEvaluation::Finger::AN2KViewVariableResolution::parse**↵  
**PositionDescriptors** ( **const** **RecordType** *typeID*, **const** **RECORD** \* *record* ) [**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

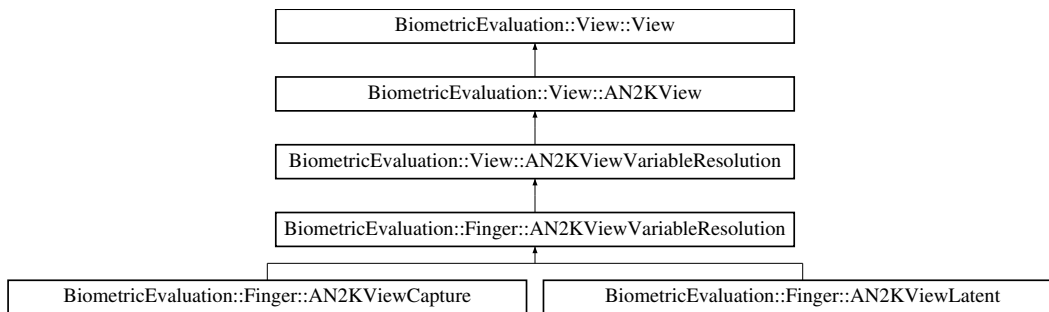
Mapping of finger position codes to finger image code.

## F.11 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

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

## Public Member Functions

- std::string [getSourceAgency](#) () const
- std::string [getCaptureDate](#) () const
- std::string [getComment](#) () const  
*Obtain the comment field.*
- [Memory::uint8Array](#) [getUserDefinedField](#) (const uint16\_t field) const  
*Obtain a user-defined field.*

## Static Public Member Functions

- static QualityMetricSet [extractQuality](#) (FIELD \*field)  
*Read a Quality Metric Set from a variable resolution AN2K record.*
- static [Memory::uint8Array](#) [parseUserDefinedField](#) (const RECORD \*const record, int fieldID)  
*Read raw bytes from a user-defined AN2K field.*

## Protected Member Functions

- [AN2KViewVariableResolution](#) (const std::string &filename, const [RecordType](#) typeID, const uint32\_t recordNumber)  
*Construct an AN2K finger view from a file.*
- [AN2KViewVariableResolution](#) ([Memory::uint8Array](#) &buf, const [RecordType](#) typeID, const uint32\_t recordNumber)  
*Construct an AN2K finger view using from a memory buffer.*
- QualityMetricSet [getQualityMetric](#) () const  
*Obtain quality metrics for associated image record.*

## Additional Inherited Members

### F.11.1 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.

### F.11.2 Constructor & Destructor Documentation

**BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution ( const std::string &filename, const RecordType typeID, const uint32\_t recordNumber ) [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.

**BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution ( Memory::uint8Array &buf, const RecordType typeID, const uint32\_t recordNumber ) [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.



### F.11.3 Member Function Documentation

**static** **QualityMetricSet** **BiometricEvaluation::View::AN2KViewVariableResolution::extractQuality** (  
**FIELD** \**field* ) [**static**]

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

## Parameters

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

## Exceptions

<a href="#"><i>Error::DataError</i></a>	The data contains an invalid value.
---	-------------------------------------

**std::string BiometricEvaluation::View::AN2KViewVariableResolution::getCaptureDate ( ) const**

## Returns

The capture date.

**std::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.

**QualityMetricSet BiometricEvaluation::View::AN2KViewVariableResolution::getQualityMetric ( ) const [protected]**

Obtain quality metrics for associated image record.

## Returns

Quality metrics

**std::string BiometricEvaluation::View::AN2KViewVariableResolution::getSourceAgency ( ) const**

## Returns

The source agency.

**Memory::uint8Array BiometricEvaluation::View::AN2KViewVariableResolution::getUserDefined↵  
Field ( const uint16\_t *field* ) const**

Obtain a user-defined field.

Fields are retrieved on-demand and then cached.

## Parameters

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

## Returns

Raw bytes read from the field.

Exceptions

<a href="#"><i>Error::ParameterError</i></a>	Invalid value for field.
--	--------------------------

**static Memory::uint8Array BiometricEvaluation::View::AN2KViewVariable↔  
Resolution::parseUserDefinedField ( const RECORD \*const record, int fieldID )  
[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

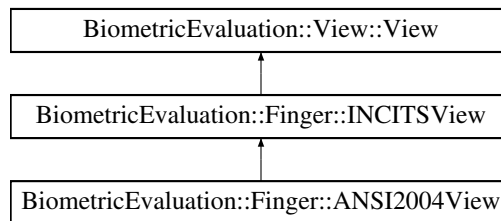
<a href="#"><i>Error::ParameterError</i></a>	Invalid value for fieldID.
--	----------------------------

## F.12 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)  
*Construct an ANSI-2004 finger view from records contained in files.*
- [\*ANSI2004View\*](#) (Memory::uint8Array &fmrBuffer, Memory::uint8Array &firBuffer, const uint32\_t↔  
viewNumber)  
*Construct an ANSI-2004 finger view from records contained in buffers.*

## Protected Member Functions

- void **readFMRHeader** ([Memory::IndexedBuffer](#) &buf)
- void **readCoreDeltaData** ([Memory::IndexedBuffer](#) &buf, uint32\_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)

*Read the core points data.*

## Static Protected Attributes

- static const uint32\_t **BASE\_SPEC\_VERSION** = 0x20323000

## Additional Inherited Members

### F.12.1 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.

### F.12.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::ANSI2004View::ANSI2004View** ( const std::string & *fmrFilename*, const std::string & *firFilename*, const uint32\_t *viewNumber* )

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.

**BiometricEvaluation::Finger::ANSI2004View::ANSI2004View** ( [Memory::uint8Array](#) & *fmrBuffer*, [Memory::uint8Array](#) & *firBuffer*, const uint32\_t *viewNumber* )

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.

### F.12.3 Member Function Documentation

**void BiometricEvaluation::Finger::ANSI2004View::readCoreDeltaData ( Memory::IndexedBuffer & buf, uint32\_t dataLength, Feature::CorePointSet & cores, Feature::DeltaPointSet & deltas )**  
**[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.

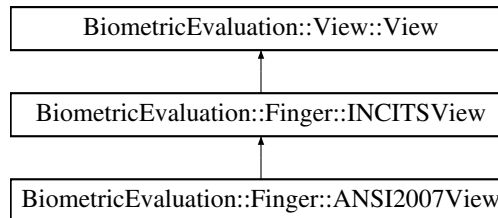
Implements [BiometricEvaluation::Finger::INCITSView](#).

## F.13 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)

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

- [ANSI2007View](#) (Memory::uint8Array &fmrBuffer, Memory::uint8Array &firBuffer, const uint32\_t↵ viewNumber)

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

### Protected Member Functions

- void **readFMRHeader** (Memory::IndexedBuffer &buf)
- void **readFVMR** (Memory::IndexedBuffer &buf)
- void **readCoreDeltaData** (Memory::IndexedBuffer &buf, uint32\_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)

*Read the core points data.*

## Static Protected Attributes

- static const uint32\_t **BASE\_SPEC\_VERSION** = 0x30333000

## Additional Inherited Members

### F.13.1 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.

### F.13.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::ANSI2007View::ANSI2007View** ( const std::string & *fmrFilename*, const std::string & *firFilename*, const uint32\_t *viewNumber* )

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

<a href="#">Error::DataError</a>	Invalid record format.
----------------------------------	------------------------

**BiometricEvaluation::Finger::ANSI2007View::ANSI2007View** ( Memory::uint8Array & *fmrBuffer*, Memory::uint8Array & *firBuffer*, const uint32\_t *viewNumber* )

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

<a href="#">Error::DataError</a>	Invalid record format.
----------------------------------	------------------------

### F.13.3 Member Function Documentation

**void BiometricEvaluation::Finger::ANSI2007View::readCoreDeltaData** ( Memory::IndexedBuffer & *buf*, uint32\_t *dataLength*, Feature::CorePointSet & *cores*, Feature::DeltaPointSet & *deltas* )  
[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.

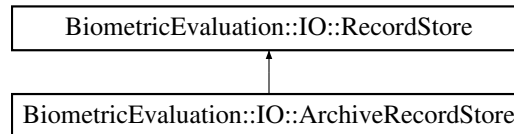
Implements [BiometricEvaluation::Finger::INCITSView](#).

## F.14 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 std::string &pathname, const std::string &description)
- [ArchiveRecordStore](#) (const std::string &pathname, uint8\_t mode=IO::READWRITE)
- [~ArchiveRecordStore](#) ()
- uint64\_t [getSpaceUsed](#) () const  
*Obtain real storage utilization.*
- void [sync](#) () const
- void [insert](#) (const std::string &key, const void \*const data, const uint64\_t size)
- void [remove](#) (const std::string &key)
- uint64\_t [read](#) (const std::string &key, void \*const data) const
- void [replace](#) (const std::string &key, const void \*const data, const uint64\_t size)
- uint64\_t [length](#) (const std::string &key) const
- void [flush](#) (const std::string &key) const
- uint64\_t [sequence](#) (std::string &key, void \*const data=nullptr, int cursor=BE\_RECSTORE\_SEQ\_NEXT)  
*Sequence through a [RecordStore](#), returning the key/data pairs.*
- void [setCursorAtKey](#) (const std::string &key)
- void [move](#) (const std::string &pathname)  
*Move the [RecordStore](#).*
- bool [needsVacuum](#) ()
- std::string [getArchiveName](#) () const
- std::string [getManifestName](#) () const
- [ArchiveRecordStore](#) (const [ArchiveRecordStore](#) &)=delete
- [ArchiveRecordStore](#) & [operator=](#) (const [ArchiveRecordStore](#) &)=delete



## Static Public Member Functions

- static bool [needsVacuum](#) (const std::string &pathname)
- static void [vacuum](#) (const std::string &pathname)

## Static Public Attributes

- static const long [OFFSET\\_RECORD\\_REMOVED](#) = -1

## Additional Inherited Members

### F.14.1 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.

### F.14.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore ( const std::string & *pathname*, const std::string & *description* )**

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

Parameters

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

Exceptions

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

**BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore ( const std::string & *pathname*, uint8\_t *mode* = *IO::READWRITE* )**

Open an existing [ArchiveRecordStore](#).

Parameters

in	<i>pathname</i>	The path name of the store.
in	<i>mode</i>	Open mode, read-only or read-write.

Exceptions

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

**BiometricEvaluation::IO::ArchiveRecordStore::~~ArchiveRecordStore ( )**

Destructor.

### F.14.3 Member Function Documentation

**void BiometricEvaluation::IO::ArchiveRecordStore::flush ( const std::string & *key* ) const**  
[**virtual**]

Commit the record's data to storage.

Parameters

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

Exceptions

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

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

**std::string BiometricEvaluation::IO::ArchiveRecordStore::getArchiveName ( ) const**

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

Returns

Path to archive file.

**std::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.

**uint64\_t BiometricEvaluation::IO::ArchiveRecordStore::getSpaceUsed ( ) const** [**virtual**]

Obtain real storage utilization.

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

Returns

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

Exceptions

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

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

**void BiometricEvaluation::IO::ArchiveRecordStore::insert ( const std::string & *key*, const void \*const *data*, const uint64\_t *size* ) [virtual]**

Insert a record into the store.

Parameters

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

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::ArchiveRecordStore::length ( const std::string & *key* ) const [virtual]**

Return the length of a record.

Parameters

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

Returns

The record length.

Exceptions

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

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

**void BiometricEvaluation::IO::ArchiveRecordStore::move ( const std::string & *pathname* ) [virtual]**

Move the [RecordStore](#).

The [RecordStore](#) can be moved to a new path in the file system.

Parameters

in	<i>pathname</i>	The new path of the <a href="#">RecordStore</a> .
----	-----------------	---

Exceptions

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

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

**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

**static bool BiometricEvaluation::IO::ArchiveRecordStore::needsVacuum ( const std::string & *pathname* ) [static]**

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

Parameters

in	<i>pathname</i>	The path name of the existing <a href="#">RecordStore</a> .
----	-----------------	---

Exceptions

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

Returns

true if [vacuum\(\)](#) would be beneficial false otherwise

**uint64\_t BiometricEvaluation::IO::ArchiveRecordStore::read ( const std::string & *key*, void \*const *data* ) const [virtual]**

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

Parameters

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

Returns

The size of the record.

Exceptions

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

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

**void BiometricEvaluation::IO::ArchiveRecordStore::remove ( const std::string & *key* ) [virtual]**

Remove a record from the store.

## Parameters

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

## Exceptions

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

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

**void BiometricEvaluation::IO::ArchiveRecordStore::replace ( const std::string & key, const void \*const data, const uint64\_t size ) [virtual]**

Replace a complete record in a store.

## Parameters

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

## Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::ArchiveRecordStore::sequence ( std::string & key, void \*const data = nullptr, int cursor = BE\_RECSTORE\_SEQ\_NEXT ) [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 sequencer 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 nullptr to indicate only the key is wanted.
in	cursor	The location within the sequence of the key/data pair to return.

## Returns

The length of the record currently in sequence.

## Exceptions

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

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

```
void BiometricEvaluation::IO::ArchiveRecordStore::setCursorAtKey ( const std::string & key )  
[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

<code>in</code>	<code>key</code>	The key of the record which will be returned by the first subsequent call to <a href="#">sequence()</a> .
-----------------	------------------	---

## Exceptions

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

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

**void BiometricEvaluation::IO::ArchiveRecordStore::sync ( ) const [virtual]**

Synchronize the entire record store to persistent storage.

## Exceptions

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

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

**static void BiometricEvaluation::IO::ArchiveRecordStore::vacuum ( const std::string &pathname ) [static]**

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

## Parameters

<code>in</code>	<code>pathname</code>	The pathname of the existing <a href="#">RecordStore</a> .
-----------------	-----------------------	--

## Exceptions

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

## Note

This is an expensive operation.

## F.14.4 Member Data Documentation

**const long BiometricEvaluation::IO::ArchiveRecordStore::OFFSET\_RECORD\_REMOVED = -1 [static]**

Offset placeholder indicating a removed record

## F.15 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

- using `value_type` = T
- using `size_type` = size\_t
- using `iterator` = `AutoArrayIterator`< false, T >
- using `const_iterator` = `AutoArrayIterator`< true, T >
- using `reference` = T &
- using `const_reference` = const T &

## 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)`  
*Subscript into the `AutoArray` with checked access.*
- `const_reference at (ptrdiff_t index) const`  
*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`.*
- `const_iterator cbegin () 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`.*
- `const_iterator cend () 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)`  
*Change the number of accessible elements.*
- `void copy (const T *buffer)`  
*Deep-copy the contents of a buffer into this `AutoArray`.*
- `void copy (const T *buffer, size_type size)`  
*Deep-copy the contents of a buffer into this `AutoArray`.*
- `AutoArray (size_type size=0)`  
*Construct an `AutoArray`.*
- `AutoArray (const AutoArray &copy)`



- Construct an [AutoArray](#).
- [AutoArray](#) ([AutoArray](#) &&rvalue) noexcept
- Construct an [AutoArray](#).
- [AutoArray](#) & operator= (const [AutoArray](#) &other)
- Copy assignment operator overload performing a deep copy.
- [AutoArray](#) & operator= ([AutoArray](#) &&other) noexcept(noexcept(std::swap(std::declval< [value\\_type](#) & >(), std::declval< [value\\_type](#) & >()))&&noexcept(std::swap(std::declval< [size\\_type](#) & >(), std::declval< [size\\_type](#) & >())))
- Move assignment operator.
- [~AutoArray](#) ()

### F.15.1 Detailed Description

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

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

### F.15.2 Member Typedef Documentation

**template<class T> using BiometricEvaluation::Memory::AutoArray< T >::const\_iterator = AutoArrayIterator<true, T>**

Const iterator of element

**template<class T> using BiometricEvaluation::Memory::AutoArray< T >::const\_reference = const T&**

Const reference element

**template<class T> using BiometricEvaluation::Memory::AutoArray< T >::iterator = AutoArrayIterator<false, T>**

Iterator of element

**template<class T> using BiometricEvaluation::Memory::AutoArray< T >::reference = T&**

Reference to element

**template<class T> using BiometricEvaluation::Memory::AutoArray< T >::size\_type = size\_t**

Type of subscripts, counts, etc.

**template<class T> using BiometricEvaluation::Memory::AutoArray< T >::value\_type = T**

Type of element

### F.15.3 Constructor & Destructor Documentation

**template<class T> BiometricEvaluation::Memory::AutoArray< T >::AutoArray ( size\_type size = 0 )**

Construct an [AutoArray](#).

Parameters

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

Exceptions

<a href="#">Error::MemoryError</a>	Could not allocate new memory.
------------------------------------	--------------------------------

**template<class T > BiometricEvaluation::Memory::AutoArray< T >::AutoArray ( const AutoArray< T > & *copy* )**

Construct an [AutoArray](#).

Parameters

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

Exceptions

<a href="#">Error::MemoryError</a>	Could not allocate new memory.
------------------------------------	--------------------------------

**template<class T > BiometricEvaluation::Memory::AutoArray< T >::AutoArray ( AutoArray< T > && *rvalue* ) [noexcept]**

Construct an [AutoArray](#).

Parameters

<i>in</i>	<i>rvalue</i>	An rvalue reference to an <a href="#">AutoArray</a> whose contents will be moved and destroyed.
-----------	---------------	---

**template<class T > BiometricEvaluation::Memory::AutoArray< T >::~~AutoArray ( )**

Destructor

## F.15.4 Member Function Documentation

**template<class T > BiometricEvaluation::Memory::AutoArray< T >::reference BiometricEvaluation::Memory::AutoArray< T >::at ( ptrdiff\_t *index* )**

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 <a href="#">AutoArray</a> .
---------------------	---

**template<class T > BiometricEvaluation::Memory::AutoArray< T >::const\_reference BiometricEvaluation::Memory::AutoArray< T >::at ( ptrdiff\_t *index* ) const**

Subscript into the [AutoArray](#) with checked access.

## Parameters

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

## Returns

Const reference to the element at the specified index.

## Exceptions

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

**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](#).

**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](#).

**template<class T > BiometricEvaluation::Memory::AutoArray< T >::const\_iterator  
BiometricEvaluation::Memory::AutoArray< T >::cbegin ( ) const**

Obtain an iterator to the beginning of the [AutoArray](#).

## Returns

Const iterator positioned at the first element of the [AutoArray](#).

**template<class T > BiometricEvaluation::Memory::AutoArray< T >::const\_iterator  
BiometricEvaluation::Memory::AutoArray< T >::cend ( ) const**

Obtain an iterator to the end of the [AutoArray](#).

## Returns

Iterator positioned at the one-past-last element of the [AutoArray](#).

**template<class T> void BiometricEvaluation::Memory::AutoArray< T >::copy ( const T \* *buffer* )**

Deep-copy the contents of a buffer into this [AutoArray](#).

Parameters

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

Warning

If buffer is smaller in size than the current size of the [AutoArray](#), you MUST call `copy(const T*, size_type, type)`. This method must only be used when buffer is larger than or equal to the size of the [AutoArray](#).

**template<class T> void BiometricEvaluation::Memory::AutoArray< T >::copy ( const T \* *buffer*, size\_type *size* )**

Deep-copy the contents of a buffer into this [AutoArray](#).

Parameters

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

Warning

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

**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](#).

**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](#).

**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.

**template<class T > BiometricEvaluation::Memory::AutoArray< T >::operator T \* ( )**

Convert [AutoArray](#) to T array.

Returns

Pointer to the beginning of the underlying array storage.

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

Copy assignment operator overload performing a deep copy.

Parameters

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

Returns

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

Exceptions

<a href="#">Error::MemoryError</a>	Could not allocate new memory.
------------------------------------	--------------------------------

```
template<class T > BiometricEvaluation::Memory::AutoArray< T > & BiometricEvaluation::Memory::AutoArray< T >::operator= ( AutoArray< T > && other )
[noexcept]
```

Move assignment operator.

Parameters

<i>in</i>	<i>other</i>	rvalue reference to another <a href="#">AutoArray</a> , whose contents will be moved and cleared from itself.
-----------	--------------	---

Returns

Reference to the lvalue [AutoArray](#).

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

Subscripting operator overload with unchecked access.

Parameters

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

Returns

Reference to the element at the specified index.

```
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

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

Returns

Const reference to the element at the specified index.

```
template<class T > void BiometricEvaluation::Memory::AutoArray< T >::resize ( size_type new_size,
bool free = false )
```

Change the number of accessible elements.

## Parameters

in	<i>new_size</i>	The number of elements the <a href="#">AutoArray</a> 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

<a href="#">Error::MemoryError</a>	Problem allocating memory.
------------------------------------	----------------------------

**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

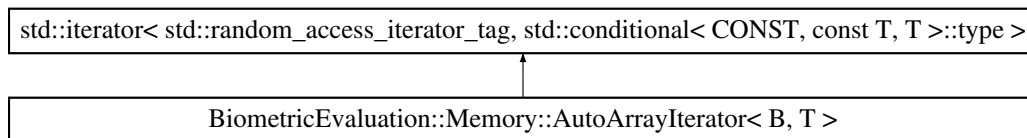
If [resize\(\)](#) has been called, the value returned from [size\(\)](#) may be smaller than the actual allocated size of the underlying storage.

## F.16 BiometricEvaluation::Memory::AutoArrayIterator< B, T > Class Template Reference

RandomAccessIterator for any [AutoArray](#).

```
#include <be_memory_autoarrayiterator.h>
```

Inheritance diagram for BiometricEvaluation::Memory::AutoArrayIterator< B, T >:



### Public Types

- using [CONTAINER](#) = typename std::conditional< CONST, const [AutoArray](#)< T > \*, [AutoArray](#)< T > \* >::type

*Convenience definition for a reference to the iterated type with appropriate constness.*

- using [POINTER](#) = typename std::conditional< CONST, const typename [AutoArrayIterator](#)< CONST, T >::pointer, typename [AutoArrayIterator](#)< CONST, T >::pointer >::type

*Convenience definition for a pointer to the iterated type with appropriate constness.*

- using [REFERENCE](#) = typename std::conditional< CONST, const typename [AutoArrayIterator](#)< CO↔NST, T >::reference, typename [AutoArrayIterator](#)< CONST, T >::reference >::type

*Convenience definition for a reference to the iterated type with appropriate constness.*

- using [DIFFERENCE](#) = typename [AutoArrayIterator](#)< CONST, T >::difference\_type

## Public Member Functions

- [AutoArrayIterator](#) ([CONTAINER](#) autoArray=nullptr, [DIFFERENCE](#) offset=0)

*Default constructor.*

- [AutoArrayIterator](#) (const [AutoArrayIterator](#) &rhs)=default
- [AutoArrayIterator](#) ([AutoArrayIterator](#) &&rhs)=default
- [~AutoArrayIterator](#) ()=default
- [AutoArrayIterator](#) & [operator=](#) ([POINTER](#) rhs)
- [AutoArrayIterator](#) & [operator=](#) (const [AutoArrayIterator](#) &rhs)=default
- [AutoArrayIterator](#) & [operator+=](#) (const [DIFFERENCE](#) &rhs)
- [AutoArrayIterator](#) & [operator-=](#) (const [DIFFERENCE](#) &rhs)
- [REFERENCE](#) [operator\\*](#) () const
- [POINTER](#) [operator->](#) () const
- [REFERENCE](#) [operator\[\]](#) (const [DIFFERENCE](#) &rhs) const
- [AutoArrayIterator](#) & [operator++](#) ()
- [AutoArrayIterator](#) & [operator--](#) ()
- [AutoArrayIterator](#) [operator++](#) (int postfix)
- [AutoArrayIterator](#) [operator--](#) (int postfix)
- [AutoArrayIterator](#) [operator+](#) (const [AutoArrayIterator](#) &rhs) const
- [DIFFERENCE](#) [operator-](#) (const [AutoArrayIterator](#)< [CONST](#), [T](#) > &rhs) const
- [AutoArrayIterator](#) [operator+](#) (const [DIFFERENCE](#) &rhs) const
- [AutoArrayIterator](#) [operator-](#) (const [DIFFERENCE](#) &rhs) const
- bool [operator==](#) (const [AutoArrayIterator](#) &rhs) const
- bool [operator!=](#) (const [AutoArrayIterator](#) &rhs) const
- bool [operator>](#) (const [AutoArrayIterator](#) &rhs) const
- bool [operator<](#) (const [AutoArrayIterator](#) &rhs) const
- bool [operator>=](#) (const [AutoArrayIterator](#) &rhs) const
- bool [operator<=](#) (const [AutoArrayIterator](#) &rhs) const

## Friends

- [AutoArrayIterator](#) [operator+](#) (const [DIFFERENCE](#) &lhs, const [AutoArrayIterator](#) &rhs)
- [AutoArrayIterator](#) [operator-](#) (const [DIFFERENCE](#) &lhs, const [AutoArrayIterator](#) &rhs)

### F.16.1 Detailed Description

**template<bool B, class T>class BiometricEvaluation::Memory::AutoArrayIterator< B, T >**

RandomAccessIterator for any [AutoArray](#).

Note

This class encapsulates a const and non-const iterator in one. The first parameter to the template is a boolean whether or not to use the const version of the iterator. The second is the contained type of the [AutoArray](#).

### F.16.2 Member Typedef Documentation

**template<bool B, class T > using BiometricEvaluation::Memory::AutoArrayIterator< B, T >::DIFFERENCE = typename AutoArrayIterator<CONST, T>::difference\_type**

Convenience definition for difference\_type



### F.16.3 Constructor & Destructor Documentation

```
template<bool B, class T > BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::AutoArrayIterator ( CONTAINER autoArray = nullptr, DIFFERENCE offset = 0 ) [inline]
```

Default constructor.

Parameters

<i>autoArray</i>	Pointer to the <a href="#">AutoArray</a> to iterate
<i>offset</i>	The offset into the <a href="#">AutoArray</a> where this iterator should start.

```
template<bool B, class T > BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::AutoArrayIterator ( const AutoArrayIterator< B, T > & rhs ) [default]
```

Default copy constructor

```
template<bool B, class T > BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::AutoArrayIterator ( AutoArrayIterator< B, T > && rhs ) [default]
```

Default move constructor

```
template<bool B, class T > BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::~~AutoArrayIterator ( ) [default]
```

Default destructor

#### F.16.4 Member Function Documentation

```
template<bool B, class T > bool BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::operator!= ( const AutoArrayIterator< B, T > & rhs ) const [inline]
```

Returns

Whether or not the offsets are different.

```
template<bool B, class T > REFERENCE BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::operator* ( ) const [inline]
```

Returns

Object at the current offset.

```
template<bool B, class T > AutoArrayIterator BiometricEvaluation::Memory::AutoArrayIterator< B,  
T >::operator+ ( const AutoArrayIterator< B, T > & rhs ) const [inline]
```

Returns

This object with offset incremented by rhs' offset.

```
template<bool B, class T > AutoArrayIterator BiometricEvaluation::Memory::AutoArrayIterator< B,  
T >::operator+ ( const DIFFERENCE & rhs ) const [inline]
```

Returns

This object with offset incremented rhs.

```
template<bool B, class T > AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator<  
B, T >::operator++ ( ) [inline]
```

Returns

This object with incremented offset.

```
template<bool B, class T > AutoArrayIterator BiometricEvaluation::Memory::AutoArrayIterator< B,  
T >::operator++ ( int postfix ) [inline]
```

Returns

This object before incrementing offset.

```
template<bool B, class T > AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator<  
B, T >::operator+= ( const DIFFERENCE & rhs ) [inline]
```

Returns

This object with rhs added to offset.

```
template<bool B, class T > DIFFERENCE BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::operator- ( const AutoArrayIterator< CONST, T > & rhs ) const [inline]
```

Returns

Offset decremented by rhs' offset.

```
template<bool B, class T > AutoArrayIterator BiometricEvaluation::Memory::AutoArrayIterator< B,  
T >::operator- ( const DIFFERENCE & rhs ) const [inline]
```

Returns

This object with offset decremented rhs.

```
template<bool B, class T > AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator<  
B, T >::operator-- ( ) [inline]
```

Returns

This object with decremented offset.

```
template<bool B, class T > AutoArrayIterator BiometricEvaluation::Memory::AutoArrayIterator< B,  
T >::operator-- ( int postfix ) [inline]
```

Returns

This object before decrementing offset.

```
template<bool B, class T > AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator<  
B, T >::operator-= ( const DIFFERENCE & rhs ) [inline]
```

Returns

This object with rhs removed from offset.

```
template<bool B, class T > POINTER BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::operator-> ( ) const [inline]
```

Returns

Address of object at the current offset.

```
template<bool B, class T > bool BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::operator< ( const AutoArrayIterator< B, T > & rhs ) const [inline]
```

Returns

true if this offset is < rhs'.

```
template<bool B, class T > bool BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::operator<= ( const AutoArrayIterator< B, T > & rhs ) const [inline]
```

Returns

true if this offset is <= rhs'.

```
template<bool B, class T > AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator<  
B, T >::operator= ( POINTER rhs ) [inline]
```

Returns

This object with offset set to rhs.

```
template<bool B, class T > AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator<  
B, T >::operator= ( const AutoArrayIterator< B, T > & rhs ) [inline], [default]
```

Default assignment operator.

```
template<bool B, class T > bool BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::operator== ( const AutoArrayIterator< B, T > & rhs ) const [inline]
```

Returns

Whether or not the offsets are the same.

```
template<bool B, class T > bool BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::operator> ( const AutoArrayIterator< B, T > & rhs ) const [inline]
```

Returns

true if this offset is > rhs'.

```
template<bool B, class T > bool BiometricEvaluation::Memory::AutoArrayIterator< B, T  
>::operator>= ( const AutoArrayIterator< B, T > & rhs ) const [inline]
```

Returns

true if this offset is >= rhs'.

```
template<bool B, class T > REFERENCE BiometricEvaluation::Memory::AutoArrayIterator< B, T
>::operator[] ( const DIFFERENCE & rhs ) const [inline]
```

Returns

Object at rhs.

### F.16.5 Friends And Related Function Documentation

```
template<bool B, class T > AutoArrayIterator operator+ ( const DIFFERENCE & lhs, const
AutoArrayIterator< B, T > & rhs ) [friend]
```

Returns

New iterator combining offsets.

```
template<bool B, class T > AutoArrayIterator operator- ( const DIFFERENCE & lhs, const
AutoArrayIterator< B, T > & rhs ) [friend]
```

Returns

New iterator differing offsets, iterating rhs' [AutoArray](#).

## F.17 BiometricEvaluation::Memory::AutoBuffer< T > Class Template Reference

### Public Types

- using [value\\_type](#) = T  
*Manage a memory buffer.*
- using **reference** = T &
- using **const\_reference** = const T &

### 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 \*)=nullptr)
- **AutoBuffer** (const [AutoBuffer](#) &copy)

#### F.17.1 Member Typedef Documentation

```
template<class T> using BiometricEvaluation::Memory::AutoBuffer< T >::value_type = T
```

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 ANSL\_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\*:

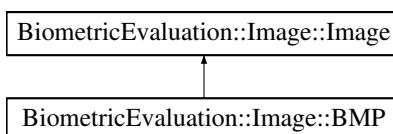
```
int size = obj->num_bytes;
```

## F.18 BiometricEvaluation::Image::BMP Class Reference

A BMP-encoded image.

```
#include <be_image_bmp.h>
```

Inheritance diagram for BiometricEvaluation::Image::BMP:



### Public Member Functions

- **BMP** (const uint8\_t \*data, const uint64\_t size)
- [Memory::AutoArray](#)< uint8\_t > [getRawData](#) () const  
*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  
*Accessor for decompressed data in grayscale.*

### Static Public Member Functions

- static bool [isBMP](#) (const uint8\_t \*data, uint64\_t size)

### Additional Inherited Members

#### F.18.1 Detailed Description

A BMP-encoded image.

Note

Only supports uncompressed BMPs with the 40-byte BITMAPINFOHEADER header information with no compression or RLE8 compression.

## F.18.2 Member Function Documentation

**Memory::AutoArray<uint8\_t> BiometricEvaluation::Image::BMP::getRawData ( ) const**  
**[virtual]**

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

Returns

AutoArray holding raw image data.

Exceptions

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

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

**Memory::AutoArray<uint8\_t> BiometricEvaluation::Image::BMP::getRawGrayscaleData ( uint8\_t depth = 8 ) const** **[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

AutoArray holding raw grayscale image data.

Exceptions

<a href="#"><i>Error::DataError</i></a>	Error decompressing image data.
<a href="#"><i>Error::ParameterError</i></a>	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

**static bool BiometricEvaluation::Image::BMP::isBMP ( const uint8\_t \* data, uint64\_t size )**  
**[static]**

Whether or not data is a [BMP](#) image.

Parameters

<i>in</i>	<i>data</i>	The buffer to check.
<i>in</i>	<i>size</i>	The size of data.

Returns

true if data appears to be a [BMP](#) image, false otherwise.

## F.19 BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet↵ Set Struct Reference

### Public Member Functions

- [CharacterSet](#) (uint16\_t [identifier](#)=0, std::string [commonName](#)="", std::string [version](#)="")

Create a new [CharacterSet](#) struct.

### Public Attributes

- uint16\_t [identifier](#)
- std::string [commonName](#)
- std::string [version](#)

### F.19.1 Constructor & Destructor Documentation

**BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::CharacterSet** ( uint16\_t *identifier* = 0, std::string *commonName* = "", std::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.

### F.19.2 Member Data Documentation

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::commonName**

Common name of the character set

**uint16\_t BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::identifier**

Identifier (000-999)

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::version**

Optional version of the character set

## F.20 BiometricEvaluation::Process::CommandCenter< T, typename >↵ ::Command Class Reference

```
#include <be.process.commandcenter.h>
```

### Public Attributes

- uint32\_t [clientID](#)
- T [command](#)
- std::vector< std::string > [arguments](#)



### F.20.1 Detailed Description

**template<typename T, typename = typename std::enable\_if<std::is\_enum<T>::value>>class BiometricEvaluation::Process::CommandCenter< T, typename >::Command**

Parsed command received from the network.

### F.20.2 Member Data Documentation

**template<typename T, typename = typename std::enable\_if<std::is\_enum<T>::value>> std::vector<std::string> BiometricEvaluation::Process::CommandCenter< T, typename >::Command::arguments**

Arguments passed to command (optional).

**template<typename T, typename = typename std::enable\_if<std::is\_enum<T>::value>> uint32\_t BiometricEvaluation::Process::CommandCenter< T, typename >::Command::clientID**

ID of the sender.

**template<typename T, typename = typename std::enable\_if<std::is\_enum<T>::value>> T BiometricEvaluation::Process::CommandCenter< T, typename >::Command::command**

Enumeration value of the command.

## F.21 BiometricEvaluation::Process::CommandCenter< T, typename > Class Template Reference

```
#include <be_process_commandcenter.h>
```

### Classes

- class [Command](#)

### Public Member Functions

- [CommandCenter](#) (uint16\_t port=[MessageCenter::DEFAULT\\_PORT](#))  
*Constructor.*
- [~CommandCenter](#) ()=default
- bool [hasPendingCommands](#) ()  
*Determine if there are commands waiting.*
- bool [getNextCommand](#) ([Command](#) &command, int numSeconds=-1, std::string invalidCommandResponse="")  
*Get the next command.*
- void [sendResponse](#) (uint32\_t clientID, const std::string &response, const std::string prefix=">> ", const std::string suffix="\n")  
*Send a string response to a client.*
- void [disconnectClient](#) (uint32\_t clientID)  
*Break the connection with a client.*

### F.21.1 Detailed Description

**template<typename T, typename = typename std::enable\_if<std::is\_enum<T>::value>>class BiometricEvaluation::Process::CommandCenter< T, typename >**

Receive enumerations as commands over the network.

### F.21.2 Constructor & Destructor Documentation

**template<typename T, typename = typename std::enable\_if<std::is\_enum<T>::value>>BiometricEvaluation::Process::CommandCenter< T, typename >::CommandCenter ( uint16\_t port = MessageCenter::DEFAULT\_PORT ) [inline]**

Constructor.

Parameters

<i>port</i>	Port to listen on for commands.
-------------	---------------------------------

**template<typename T, typename = typename std::enable\_if<std::is\_enum<T>::value>>BiometricEvaluation::Process::CommandCenter< T, typename >::~~CommandCenter ( ) [default]**

Destructor (default).

### F.21.3 Member Function Documentation

**template<typename T, typename = typename std::enable\_if<std::is\_enum<T>::value>> void BiometricEvaluation::Process::CommandCenter< T, typename >::disconnectClient ( uint32\_t clientID ) [inline]**

Break the connection with a client.

Parameters

<i>clientID</i>	ID of the client to disconnect.
-----------------	---------------------------------

**template<typename T, typename = typename std::enable\_if<std::is\_enum<T>::value>> bool BiometricEvaluation::Process::CommandCenter< T, typename >::getNextCommand ( Command & command, int numSeconds = -1, std::string invalidCommandResponse = "" ) [inline]**

Get the next command.

Parameters

<i>command</i>	Reference to a <a href="#">Command</a> that will be populated when this method returns true.
<i>numSeconds</i>	Number of seconds to wait for a command, or -1 to block indefinitely.
<i>invalidCommandResponse</i>	Optional string to send, such as usage, that will be sent when an unrecognized command is received.

Returns

true if command has been populated, false otherwise.

```
template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> bool
BiometricEvaluation::Process::CommandCenter< T, typename >::hasPendingCommands ( )
[inline]
```

Determine if there are commands waiting.

Returns

true if there are commands waiting, false otherwise.

Note

Returns immediately.

See also

[BiometricEvaluation::Process::CommandCenter::getNextCommand\(\)](#)

```
template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> void
BiometricEvaluation::Process::CommandCenter< T, typename >::sendResponse ( uint32_t clientID,
const std::string & response, const std::string prefix = ">> ", const std::string suffix = "\n" )
[inline]
```

Send a string response to a client.

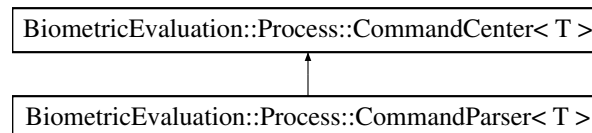
Parameters

<i>clientID</i>	ID of client to communicate with.
<i>response</i>	Printable string to send to client.
<i>prefix</i>	String to prefix to responses.
<i>suffix</i>	String to append to responses.

## F.22 BiometricEvaluation::Process::CommandParser< T > Class Template Reference

```
#include <be_process_commandcenter.h>
```

Inheritance diagram for BiometricEvaluation::Process::CommandParser< T >:



### Public Member Functions

- virtual void [parse](#) (const typename [CommandCenter](#)< T >::Command &command)=0  
*Parse command.*
- bool [getNextCommand](#) (typename [CommandCenter](#)< T >::Command &command, int numSeconds=-1)  
*Get the next command.*
- void [setUsage](#) (const std::string &usage)

*String sent when an invalid command is received.*

- std::string `getUsage ()` const
- `CommandParser` (uint16\_t port=`MessageCenter::DEFAULT_PORT`)

*Constructor.*

- virtual `~CommandParser ()`=default

### F.22.1 Detailed Description

**template<typename T>class BiometricEvaluation::Process::CommandParser< T >**

Abstraction to parse messages received via `CommandCenter`.

### F.22.2 Constructor & Destructor Documentation

**template<typename T > BiometricEvaluation::Process::CommandParser< T >::CommandParser ( uint16\_t port = MessageCenter::DEFAULT\_PORT ) [inline]**

Constructor.

Parameters

<i>port</i>	Port to listen on for commands.
-------------	---------------------------------

**template<typename T > virtual BiometricEvaluation::Process::CommandParser< T >::~~CommandParser ( ) [virtual], [default]**

Virtual destructor (default).

### F.22.3 Member Function Documentation

**template<typename T > bool BiometricEvaluation::Process::CommandParser< T >::getNextCommand ( typename CommandCenter< T >::Command & command, int numSeconds = -1 ) [inline]**

Get the next command.

Parameters

<i>command</i>	Reference to a Command that will be populated when this method returns true.
<i>numSeconds</i>	Number of seconds to wait for a command, or -1 to block indefinitely.

Returns

true if command has been populated, false otherwise.

**template<typename T > std::string BiometricEvaluation::Process::CommandParser< T >::getUsage ( ) const [inline]**

Returns

Usage string.

```
template<typename T > virtual void BiometricEvaluation::Process::CommandParser< T >::parse (  
const typename CommandCenter< T >::Command & command ) [pure virtual]
```

Parse command.

Implement this method as a switch statement of your command enumeration.

```
template<typename T > void BiometricEvaluation::Process::CommandParser< T >::setUsage ( const  
std::string & usage ) [inline]
```

String sent when an invalid command is received.

## Parameters

<i>usage</i>	String to send when an invalid command is received.
--------------	---

## Note

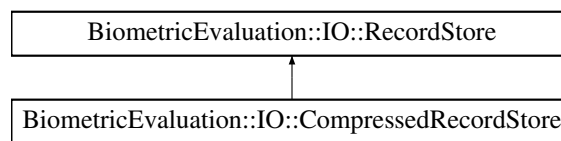
If not set, no additional usage is sent.

## F.23 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 std::string &pathname, const std::string &description, const [RecordStore::Kind](#) &recordStoreType, const std::string &compressorType)
- [CompressedRecordStore](#) (const std::string &pathname, const std::string &description, const [RecordStore::Kind](#) &recordStoreType, const [Compressor::Kind](#) &compressorType)
- [CompressedRecordStore](#) (const std::string &pathname, uint8\_t mode=IO::READWRITE)
- uint64\_t [getSpaceUsed](#) () const  
*Obtain real storage utilization.*
- void [sync](#) () const
- void [insert](#) (const std::string &key, const void \*const data, const uint64\_t size)
- void [remove](#) (const std::string &key)
- uint64\_t [read](#) (const std::string &key, void \*const data) const
- void [replace](#) (const std::string &key, const void \*const data, const uint64\_t size)
- uint64\_t [length](#) (const std::string &key) const
- void [flush](#) (const std::string &key) const
- uint64\_t [sequence](#) (std::string &key, void \*const data=nullptr, int cursor=BE\_RECSTORE\_SEQ\_NEXT)  
*Sequence through a [RecordStore](#), returning the key/data pairs.*
- void [setCursorAtKey](#) (const std::string &key)
- void [move](#) (const std::string &pathname)  
*Move the [RecordStore](#).*
- [CompressedRecordStore](#) (const [CompressedRecordStore](#) &rhs)=delete  
*Copy constructor (disabled).*
- [CompressedRecordStore](#) & [operator=](#) (const [CompressedRecordStore](#) &rhs)=delete  
*Assignment operator (disabled).*

### Static Public Attributes

- static const std::string [BACKING\\_STORE](#)
- static const std::string [COMPRESSOR\\_TYPE\\_KEY](#)

## Additional Inherited Members

### F.23.1 Detailed Description

Sibling-implemented [RecordStore](#) with Compression.

### F.23.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore ( const std::string & *pathname*, const std::string & *description*, const RecordStore::Kind & *recordStoreType*, const std::string & *compressorType* )**

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

Parameters

in	<i>pathname</i>	The directory where the store is to be created.
in	<i>description</i>	The store's description.
in	<i>recordStoreType</i>	The type of <a href="#">RecordStore</a> subclass the internal RecordStores should be.
in	<i>compressorType</i>	The type of compression that should be used within the internal Record↔Stores.

Exceptions

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

**BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore ( const std::string & *pathname*, const std::string & *description*, const RecordStore::Kind & *recordStoreType*, const Compressor::Kind & *compressorType* )**

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

Parameters

in	<i>pathname</i>	The directory where the store is to be created.
in	<i>description</i>	The store's description.
in	<i>recordStoreType</i>	The type of <a href="#">RecordStore</a> subclass the internal RecordStores should be.
in	<i>compressorType</i>	The type of compression that should be used within the internal Record↔Stores.

Exceptions

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

**BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore ( const std::string & *pathname*, uint8\_t *mode* = IO::READWRITE )**

Open an existing [CompressedRecordStore](#).

Parameters

in	<i>pathname</i>	The path name of the store.
in	<i>mode</i>	Open mode, read-only or read-write.

Exceptions

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

**BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore ( const CompressedRecordStore & rhs ) [delete]**

Copy constructor (disabled).

Disabled because this object could represent a file on disk.

Parameters

<i>rhs</i>	<a href="#">CompressedRecordStore</a> object to copy.
------------	---

### F.23.3 Member Function Documentation

**void BiometricEvaluation::IO::CompressedRecordStore::flush ( const std::string & key ) const [virtual]**

Commit the record's data to storage.

Parameters

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

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::CompressedRecordStore::getSpaceUsed ( ) const [virtual]**

Obtain real storage utilization.

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

Returns

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

Exceptions

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

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

**void BiometricEvaluation::IO::CompressedRecordStore::insert ( const std::string & key, const void \*const data, const uint64\_t size ) [virtual]**

Insert a record into the store.



## Parameters

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

## Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::CompressedRecordStore::length ( const std::string & *key* ) const**  
**[virtual]**

Return the length of a record.

## Parameters

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

## Returns

The record length.

## Exceptions

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

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

**void BiometricEvaluation::IO::CompressedRecordStore::move ( const std::string & *pathname* )**  
**[virtual]**

Move the [RecordStore](#).

The [RecordStore](#) can be moved to a new path in the file system.

## Parameters

in	<i>pathname</i>	The new path of the <a href="#">RecordStore</a> .
----	-----------------	---

## Exceptions

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

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

**CompressedRecordStore& BiometricEvaluation::IO::CompressedRecordStore::operator= ( const**  
**CompressedRecordStore & *rhs* ) [delete]**

Assignment operator (disabled).

Disabled because this object could represent a file on disk.

## Parameters

<i>rhs</i>	<a href="#">CompressedRecordStore</a> object to assign.
------------	---

Returns

[CompressedRecordStore](#) object, now containing the contents of rhs.

**uint64\_t BiometricEvaluation::IO::CompressedRecordStore::read ( const std::string & key, void \*const data ) const** [**virtual**]

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

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

Returns

The size of the record.

Exceptions

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

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

**void BiometricEvaluation::IO::CompressedRecordStore::remove ( const std::string & key )** [**virtual**]

Remove a record from the store.  
Parameters

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

Exceptions

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

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

**void BiometricEvaluation::IO::CompressedRecordStore::replace ( const std::string & key, const void \*const data, const uint64\_t size )** [**virtual**]

Replace a complete record in a store.  
Parameters

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

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::CompressedRecordStore::sequence ( std::string & key, void \*const data = nullptr, int cursor = BE\_RECSTORE\_SEQ\_NEXT ) [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 nullptr to indicate only the key is wanted.
in	cursor	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

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

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

**void BiometricEvaluation::IO::CompressedRecordStore::setCursorAtKey ( const std::string & key ) [virtual]**

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

Parameters

in	key	The key of the record which will be returned by the first subsequent call to <a href="#">sequence()</a> .
----	-----	---

Exceptions

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

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

**void BiometricEvaluation::IO::CompressedRecordStore::sync ( ) const [virtual]**

Synchronize the entire record store to persistent storage.

Exceptions

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

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

## F.23.4 Member Data Documentation

**const std::string BiometricEvaluation::IO::CompressedRecordStore::BACKING\_STORE [static]**

Name of the underlying store within this RS

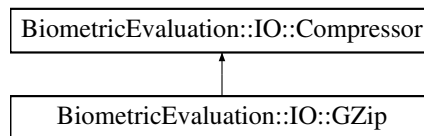
```
const std::string BiometricEvaluation::IO::CompressedRecordStore::COMPRESSOR_TYPE_KEY
[static]
```

Name of the key storing compressor type

## F.24 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  
*Compress a buffer.*
- virtual [Memory::uint8Array compress](#) (const [Memory::uint8Array](#) &uncompressedData) const =0  
*Compress a buffer.*
- virtual void [compress](#) (const uint8\_t \*const uncompressedData, uint64\_t uncompressedDataSize, const std::string &outputFile) const =0  
*Compress a buffer.*
- virtual void [compress](#) (const [Memory::uint8Array](#) &uncompressedData, const std::string &outputFile) const =0  
*Compress a buffer.*
- virtual [Memory::uint8Array compress](#) (const std::string &inputFile) const =0  
*Compress a file.*
- virtual void [compress](#) (const std::string &inputFile, const std::string &outputFile) const =0  
*Compress a file.*
- virtual [Memory::uint8Array decompress](#) (const uint8\_t \*const compressedData, uint64\_t compressedDataSize) const =0  
*Decompress a compressed buffer.*
- virtual [Memory::uint8Array decompress](#) (const [Memory::uint8Array](#) &compressedData) const =0  
*Decompress a compressed buffer.*
- virtual [Memory::uint8Array decompress](#) (const std::string &inputFile) const =0  
*Decompress a compressed buffer into a file.*
- virtual void [decompress](#) (const [Memory::uint8Array](#) &compressedData, const std::string &outputFile) const =0

- Decompress a file.*
- virtual void [decompress](#) (const uint8\_t \*const compressedData, const uint64\_t compressedDataSize, const std::string &outputFile) const =0
- Decompress a file.*
- virtual void [decompress](#) (const std::string &inputFile, const std::string &outputFile) const =0
- Decompress a file.*
- void [setOption](#) (const std::string &optionName, const std::string &optionValue)
- Assign a compressor option.*
- void [setOption](#) (const std::string &optionName, int64\_t optionValue)
- Assign a compressor option.*
- std::string [getOption](#) (const std::string &optionName) const
- Obtain a compressor option as an integer.*
- int64\_t [getOptionAsInteger](#) (const std::string &optionName) const
- Obtain a compressor option as an integer.*
- void [removeOption](#) (const std::string &optionName)
- Remove a compressor option.*
- virtual [~Compressor](#) ()
- [Compressor](#) (const [Compressor](#) &other)=delete
- Copy constructor (disabled).*
- [Compressor](#) & [operator=](#) (const [Compressor](#) &other)=delete
- Assignment overload (disabled).*

## Static Public Member Functions

- static std::shared\_ptr  
< [Compressor](#) > [createCompressor](#) ([Compressor::Kind](#) compressorKind=[Kind::GZIP](#))

### F.24.1 Detailed Description

Implementations for compressing and decompressing data

### F.24.2 Member Enumeration Documentation

**enum BiometricEvaluation::IO::Compressor::Kind [strong]**

Kinds of Compressors (for factory)

### F.24.3 Constructor & Destructor Documentation

**BiometricEvaluation::IO::Compressor::Compressor ( )**

Create a new [Compressor](#) object.

Default compression options will be used.

**virtual BiometricEvaluation::IO::Compressor::~~Compressor ( ) [virtual]**

Destructor

**BiometricEvaluation::IO::Compressor::Compressor ( const [Compressor](#) & *other* ) [delete]**

Copy constructor (disabled).

Disabled because [Properties](#) member cannot be copied.

Parameters

<i>other</i>	<a href="#">Compressor</a> to copy.
--------------	-------------------------------------

## F.24.4 Member Function Documentation

**virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress ( const uint8\_t \*const *uncompressedData*, uint64\_t *uncompressedDataSize* ) const** **[pure virtual]**

Compress a buffer.

Parameters

<i>uncompressedData</i>	Uncompressed data buffer to compress.
<i>uncompressedDataSize</i>	Size of uncompressedData.

Returns

Compressed buffer.

Exceptions

<a href="#">Error::StrategyError</a>	<a href="#">Error</a> in compression unit.
--------------------------------------	--

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

**virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress ( const Memory::uint8Array & *uncompressedData* ) const** **[pure virtual]**

Compress a buffer.

Parameters

<i>uncompressedData</i>	Uncompressed data buffer to compress.
-------------------------	---------------------------------------

Returns

Compressed buffer.

Exceptions

<a href="#">Error::StrategyError</a>	<a href="#">Error</a> in decompression unit.
--------------------------------------	--

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

**virtual void BiometricEvaluation::IO::Compressor::compress ( const uint8\_t \*const *uncompressedData*, uint64\_t *uncompressedDataSize*, const std::string & *outputFile* ) const** **[pure virtual]**

Compress a buffer.

Parameters

<i>uncompressedData</i>	Uncompressed data buffer to compress.
<i>uncompressedDataSize</i>	Size of uncompressedData.
<i>outputFile</i>	Location to save compressed file.

Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in compression unit.

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

**virtual void BiometricEvaluation::IO::Compressor::compress ( const Memory::uint8Array & *uncompressedData*, const std::string & *outputFile* ) const [pure virtual]**

Compress a buffer.

Parameters

<i>uncompressedData</i>	Uncompressed data buffer to compress.
<i>outputFile</i>	Location to save compressed file.

Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in decompression unit.

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

**virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress ( const std::string & *inputFile* ) const [pure virtual]**

Compress a file.

Parameters

<i>inputFile</i>	Path to file to compress.
------------------	---------------------------

Returns

Compressed buffer.

Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	Input file does not exist.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in decompression unit.

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

**virtual void BiometricEvaluation::IO::Compressor::compress ( const std::string & *inputFile*, const std::string & *outputFile* ) const [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

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	Input file does not exist.
<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in decompression unit.

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

**static std::shared\_ptr<Compressor> BiometricEvaluation::IO::Compressor::createCompressor ( Compressor::Kind *compressorKind* = *Kind::GZIP* ) [static]**

[Compressor](#) factory.

Parameters

<i>compressorKind</i>	A known kind of compressor.
-----------------------	-----------------------------

Returns

A new compressor with default options.

Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	Invalid compressor type.
--	--------------------------

**virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress ( const uint8\_t \*const *compressedData*, uint64\_t *compressedDataSize* ) const [pure virtual]**

Decompress a compressed buffer.

Parameters

<i>compressedData</i>	Compressed data buffer to decompress.
<i>compressedDataSize</i>	Size of <i>compressedData</i> .

Returns

Decompressed data.

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in compression unit.
---	--

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

**virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress ( const Memory::uint8Array & *compressedData* ) const [pure virtual]**

Decompress a compressed buffer.



## Parameters

<i>compressed</i> ↔ <i>Data</i>	Compressed data buffer to decompress.
------------------------------------	---------------------------------------

## Returns

Decompressed data.

## Exceptions

<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in decompression unit.
---	--

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

**virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress ( const std::string & *inputFile* ) const [pure virtual]**

Decompress a compressed buffer into a file.

## Parameters

<i>inputFile</i>	Location to save compressed file.
------------------	-----------------------------------

## Returns

Decompressed data.

## Exceptions

<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in decompression unit.
<i>Error::ObjectDoesNot</i> ↔ <i>Exists</i>	Output file already exists.

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

**virtual void BiometricEvaluation::IO::Compressor::decompress ( const Memory::uint8Array & *compressedData*, const std::string & *outputFile* ) const [pure virtual]**

Decompress a file.

## Parameters

<i>compressed</i> ↔ <i>Data</i>	Compressed data buffer to decompress.
<i>outputFile</i>	Path to location where decompressed version will be saved.

## Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in compression unit.

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

**virtual void BiometricEvaluation::IO::Compressor::decompress ( const uint8\_t \*const *compressedData*, const uint64\_t *compressedDataSize*, const std::string & *outputFile* ) const [pure virtual]**

Decompress a file.

Parameters

<i>compressedData</i>	Compressed data buffer to decompress.
<i>compressedDataSize</i>	Size of compressedData.
<i>outputFile</i>	Path to location where decompressed version will be saved.

Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in compression unit.

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

**virtual void BiometricEvaluation::IO::Compressor::decompress ( const std::string & *inputFile*, const std::string & *outputFile* ) const** **[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

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	Input file does not exist.
<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in compression unit.

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

**std::string BiometricEvaluation::IO::Compressor::getOption ( const std::string & *optionName* ) const**

Obtain a compressor option as an integer.

Parameters

<i>optionName</i>	Name of the option to obtain.
-------------------	-------------------------------

Returns

Value of compressor option.

**int64\_t BiometricEvaluation::IO::Compressor::getOptionAsInteger ( const std::string & *optionName* ) const**

Obtain a compressor option as an integer.

Parameters

<i>optionName</i>	Name of the option to obtain.
-------------------	-------------------------------

Returns

Value of compressor option.

## Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The option was never set.
--	---------------------------

### Compressor & BiometricEvaluation::IO::Compressor::operator= ( const Compressor & *other* ) [delete]

Assignment overload (disabled).

Disabled because [Properties](#) member cannot be assigned.

## Parameters

<i>other</i>	<a href="#">Compressor</a> to assign.
--------------	---------------------------------------

## Returns

lhs [Compressor](#).

### void BiometricEvaluation::IO::Compressor::removeOption ( const std::string & *optionName* )

Remove a compressor option.

## Parameters

<i>optionName</i>	Name of the option to remove.
-------------------	-------------------------------

### void BiometricEvaluation::IO::Compressor::setOption ( const std::string & *optionName*, const std::string & *optionValue* )

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

<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> setting option.
---	---------------------------------------

### void BiometricEvaluation::IO::Compressor::setOption ( const std::string & *optionName*, int64\_t *optionValue* )

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

<a href="#">Error::StrategyError</a>	<a href="#">Error</a> setting option.
--------------------------------------	---------------------------------------

## F.25 BiometricEvaluation::Framework::ConstEnumMapWrapper< T > Class Template Reference

Wrapper class around an individual enumeration entity (const).

```
#include <be_framework_enumeration.h>
```

### Public Member Functions

- [ConstEnumMapWrapper](#) (const T &enumeration)
- [operator std::string](#) () const
- [operator T](#) () const

#### F.25.1 Detailed Description

**template<typename T>class BiometricEvaluation::Framework::ConstEnumMapWrapper< T >**

Wrapper class around an individual enumeration entity (const).

Because the operators are in the main namespace for maximum usefulness, we must create this additional type to avoid type ambiguity when using more than one template (e.g., string) in a source file.

#### F.25.2 Constructor & Destructor Documentation

**template<typename T > BiometricEvaluation::Framework::ConstEnumMapWrapper< T >::ConstEnumMapWrapper ( const T & *enumeration* )**

Constructor

#### F.25.3 Member Function Documentation

**template<typename T > BiometricEvaluation::Framework::ConstEnumMapWrapper< T >::operator std::string ( ) const**

Implicit conversion to std::string

**template<typename T > BiometricEvaluation::Framework::ConstEnumMapWrapper< T >::operator T ( ) const**

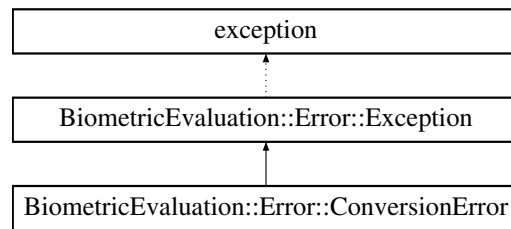
Implicit conversion to enumeration

## F.26 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](#) (std::string info)

### F.26.1 Detailed Description

[Error](#) when converting one object into another, a property value from string to int, for example.

### F.26.2 Constructor & Destructor Documentation

**[BiometricEvaluation::Error::ConversionError::ConversionError](#) ( )**

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

**[BiometricEvaluation::Error::ConversionError::ConversionError](#) ( std::string *info* )**

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

## F.27 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

### F.27.1 Detailed Description

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

### F.27.2 Constructor & Destructor Documentation

**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	<i>x</i>	X-coordinate
in	<i>y</i>	Y-coordinate
in	<i>xDistance</i>	X-coordinate distance from origin
in	<i>yDistance</i>	Y-coordinate distance from origin

### F.27.3 Member Data Documentation

**uint32\_t BiometricEvaluation::Image::Coordinate::x**

X-coordinate

**float BiometricEvaluation::Image::Coordinate::xDistance**

X-coordinate distance from origin

**uint32\_t BiometricEvaluation::Image::Coordinate::y**

Y-coordinate

**float BiometricEvaluation::Image::Coordinate::yDistance**

Y-coordinate distance from origin

## F.28 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**

### F.28.1 Detailed Description

Representation of the core.

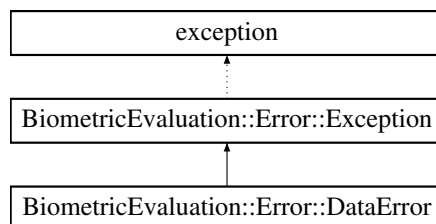
A core has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

## F.29 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](#) (std::string info)

### F.29.1 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.

### F.29.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::DataError::DataError ( )**

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

**BiometricEvaluation::Error::DataError::DataError ( std::string *info* )**

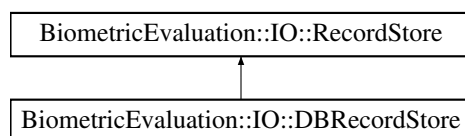
Construct a [DataError](#) object with an information string appended to the default information string.

## F.30 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 std::string &pathname, const std::string &description)
- [DBRecordStore](#) (const std::string &pathname, uint8\_t mode=IO::READWRITE)
- uint64\_t [getSpaceUsed](#) () const  
*Obtain real storage utilization.*
- void [sync](#) () const
- void [insert](#) (const std::string &key, const void \*const data, const uint64\_t size)
- void [remove](#) (const std::string &key)
- uint64\_t [read](#) (const std::string &key, void \*const data) const
- void [replace](#) (const std::string &key, const void \*const data, const uint64\_t size)
- uint64\_t [length](#) (const std::string &key) const
- void [flush](#) (const std::string &key) const
- uint64\_t [sequence](#) (std::string &key, void \*const data=nullptr, int cursor=BE\_RECSTORE\_SEQ\_NEXT)  
*Sequence through a [RecordStore](#), returning the key/data pairs.*
- void [setCursorAtKey](#) (const std::string &key)
- void [move](#) (const std::string &pathname)  
*Move the [RecordStore](#).*
- [DBRecordStore](#) (const [DBRecordStore](#) &)=delete
- [DBRecordStore](#) & [operator=](#) (const [DBRecordStore](#) &)=delete

## Additional Inherited Members

### F.30.1 Detailed Description

A class that implements [IO::RecordStore](#) using a Berkeley DB database as the underlying record storage system.

### F.30.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::DBRecordStore::DBRecordStore ( const std::string & *pathname*, const std::string & *description* )**

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

Parameters

in	<i>pathname</i>	The directory where the store will be created.
in	<i>description</i>	The store's description.

Exceptions

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

**BiometricEvaluation::IO::DBRecordStore::DBRecordStore ( const std::string & *pathname*, uint8\_t *mode* = *IO::READWRITE* )**

Open an existing [DBRecordStore](#).

Parameters

in	<i>name</i>	The path name of the store.
in	<i>mode</i>	Open mode, read-only or read-write.

Exceptions

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

### F.30.3 Member Function Documentation

**void BiometricEvaluation::IO::DBRecordStore::flush ( const std::string & *key* ) const [virtual]**

Commit the record's data to storage.

Parameters

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

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::DBRecordStore::getSpaceUsed ( ) const [virtual]**

Obtain real storage utilization.

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

Returns

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

Exceptions

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

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

**void BiometricEvaluation::IO::DBRecordStore::insert ( const std::string & *key*, const void \*const *data*, const uint64\_t *size* ) [virtual]**

Insert a record into the store.

Parameters

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

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::DBRecordStore::length ( const std::string & *key* ) const**  
**[virtual]**

Return the length of a record.

Parameters

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

Returns

The record length.

Exceptions

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

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

**void BiometricEvaluation::IO::DBRecordStore::move ( const std::string & *pathname* )** **[virtual]**

Move the [RecordStore](#).

The [RecordStore](#) can be moved to a new path in the file system.

Parameters

in	<i>pathname</i>	The new path of the <a href="#">RecordStore</a> .
----	-----------------	---

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::DBRecordStore::read ( const std::string & *key*, void \*const *data* )**  
**const** **[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><a href="#">Error::ObjectDoesNotExist</a></i>	A record for the key does not exist.
<i><a href="#">Error::StrategyError</a></i>	An error occurred when using the underlying storage system.

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

**void BiometricEvaluation::IO::DBRecordStore::remove ( const std::string &key ) [virtual]**

Remove a record from the store.

Parameters

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

Exceptions

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

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

**void BiometricEvaluation::IO::DBRecordStore::replace ( const std::string &key, const void \*const data, const uint64\_t size ) [virtual]**

Replace a complete record in a store.

Parameters

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

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::DBRecordStore::sequence ( std::string &key, void \*const data = nullptr, int cursor = BE\_RECSTORE\_SEQ\_NEXT ) [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 sequencer 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 nullptr to indicate only the key is wanted.
in	cursor	The location within the sequence of the key/data pair to return.

## Returns

The length of the record currently in sequence.

## Exceptions

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

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

**void BiometricEvaluation::IO::DBRecordStore::setCursorAtKey ( const std::string & key )**  
**[virtual]**

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

## Parameters

in	key	The key of the record which will be returned by the first subsequent call to <a href="#">sequence()</a> .
----	-----	---

## Exceptions

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

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

**void BiometricEvaluation::IO::DBRecordStore::sync ( ) const** **[virtual]**

Synchronize the entire record store to persistent storage.

## Exceptions

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

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

## F.31 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

### F.31.1 Detailed Description

Representation of the delta.

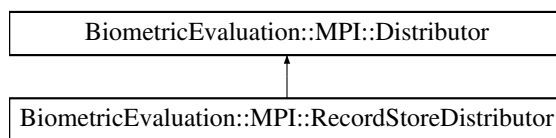
A delta has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

## F.32 BiometricEvaluation::MPI::Distributor Class Reference

A class to represent an [MPI](#) task that distributes work to other tasks.

```
#include <be_mpi_distributor.h>
```

Inheritance diagram for BiometricEvaluation::MPI::Distributor:



### Public Member Functions

- [Distributor](#) (const std::string &propertiesFileName)

*Constructor with properties file name.*

- void [start](#) ()

*Start of [MPI](#) processing for the distributor.*

### Protected Member Functions

- virtual void [createWorkPackage](#) ([MPI::WorkPackage](#) &workPackage)=0

*Create a work package for distribution.*

- std::shared\_ptr< [IO::Logsheet](#) > [getLogsheet](#) () const

*Get access to the Logsheet object.*

### F.32.1 Detailed Description

A class to represent an [MPI](#) task that distributes work to other tasks.

A [Distributor](#) object is based on a set of properties contained in a file. This class must be subclassed and an implementation of the [createWorkPackage\(\)](#) method provided.

The distributor sends an [MPI](#) message to each receiver object indicating whether it should start and ready for accepting work packages, or proceed immediately to the shutdown state. Failure to start the [Distributor](#) object will result in the entire [MPI](#) job shutting down before any work is done.

If the Logsheet URL property is set, log messages will be written to that sheet. Otherwise, log messages will be written to a Null Logsheet.

See also

[IO::Properties](#)

[MPI::Receiver](#)

[MPI::WorkPackage](#)

### F.32.2 Constructor & Destructor Documentation

**BiometricEvaluation::MPI::Distributor::Distributor** ( `const std::string & propertiesFileName` )

Constructor with properties file name.

## Parameters

in	<i>propertiesFile</i> ↔ <i>Name</i>	The name of the file containing the properties for the new object.
----	--	--

## Exceptions

<i>Error::Exception</i>	An error occurred, possibly due to missing or invalid properties.
-------------------------	---

**F.32.3 Member Function Documentation**

**virtual void BiometricEvaluation::MPI::Distributor::createWorkPackage ( MPI::WorkPackage & workPackage ) [protected], [pure virtual]**

Create a work package for distribution.

Implementations of this class create a work package to encapsulate the specific data type that is to be distributed.

Implemented in [BiometricEvaluation::MPI::RecordStoreDistributor](#).

**std::shared\_ptr<IO::Logsheet> BiometricEvaluation::MPI::Distributor::getLogsheet ( ) const [protected]**

Get access to the Logsheet object.

## Returns

A shared pointer for the Logsheet object.

**void BiometricEvaluation::MPI::Distributor::start ( )**

Start of [MPI](#) processing for the distributor.

Once started, the distributor will send a message to each receiver task telling it to start and waiting for status back from each receiver.

**F.33 BiometricEvaluation::DataInterchange::AN2KRecord::Domain↔  
Name 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](#) (std::string [identifier](#)="", std::string [version](#)="")

Create a [DomainName](#) struct.

**Public Attributes**

- std::string [identifier](#)
- std::string [version](#)

**F.33.1 Detailed Description**

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



### F.33.2 Constructor & Destructor Documentation

**BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::DomainName** ( `std::string` *identifier* = "", `std::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.

### F.33.3 Member Data Documentation

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::identifier**

Unique identifier for agency, entity, or implementation.

**std::string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::version**

Optional version of the implementation

## F.34 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification↔ ::Entry Struct Reference

### Public Member Functions

- [Entry](#) (bool [standard](#), std::string [code](#))

### Public Attributes

- bool [standard](#)
- std::string [code](#)

### F.34.1 Constructor & Destructor Documentation

**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.

### F.34.2 Member Data Documentation

**std::string BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::code**

AN2K or user-defined pattern classification code.

**bool BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::standard**

Whether code is a standard AN2K pattern classification code.

## F.35 BiometricEvaluation::Framework::EnumerationFunctions< T > Class Template Reference

```
#include <be_framework_enumeration.h>
```

## Static Public Attributes

- static const std::map< T, std::string > [enumToStringMap](#)

### F.35.1 Detailed Description

**template<typename T>class BiometricEvaluation::Framework::EnumerationFunctions< T >**

Class to store enumeration/string mappings.

### F.35.2 Member Data Documentation

**template<typename T > const std::map<T, std::string> BiometricEvaluation::Framework::↵ EnumerationFunctions< T >::enumToStringMap [static]**

Enumeration -> String Representation

## F.36 BiometricEvaluation::Framework::EnumMapWrapper< T > Class Template Reference

Wrapper class around an individual enumeration entity (non-const).

```
#include <be_framework_enumeration.h>
```

## Public Member Functions

- [EnumMapWrapper](#) (T &enumeration)
- [operator std::string](#) ()
- [operator T](#) ()

### F.36.1 Detailed Description

**template<typename T>class BiometricEvaluation::Framework::EnumMapWrapper< T >**

Wrapper class around an individual enumeration entity (non-const).

Because the operators are in the main namespace for maximum usefulness, we must create this additional type to avoid type ambiguity when using more than one template (e.g., string) in a source file.

### F.36.2 Constructor & Destructor Documentation

**template<typename T > BiometricEvaluation::Framework::EnumMapWrapper< T >::EnumMapWrapper ( T & *enumeration* )**

Constructor

### F.36.3 Member Function Documentation

**template<typename T > BiometricEvaluation::Framework::EnumMapWrapper< T >::operator std::string ( )**

Implicit conversion to std::string

```
template<typename T > BiometricEvaluation::Framework::EnumMapWrapper< T >::operator T (
)
```

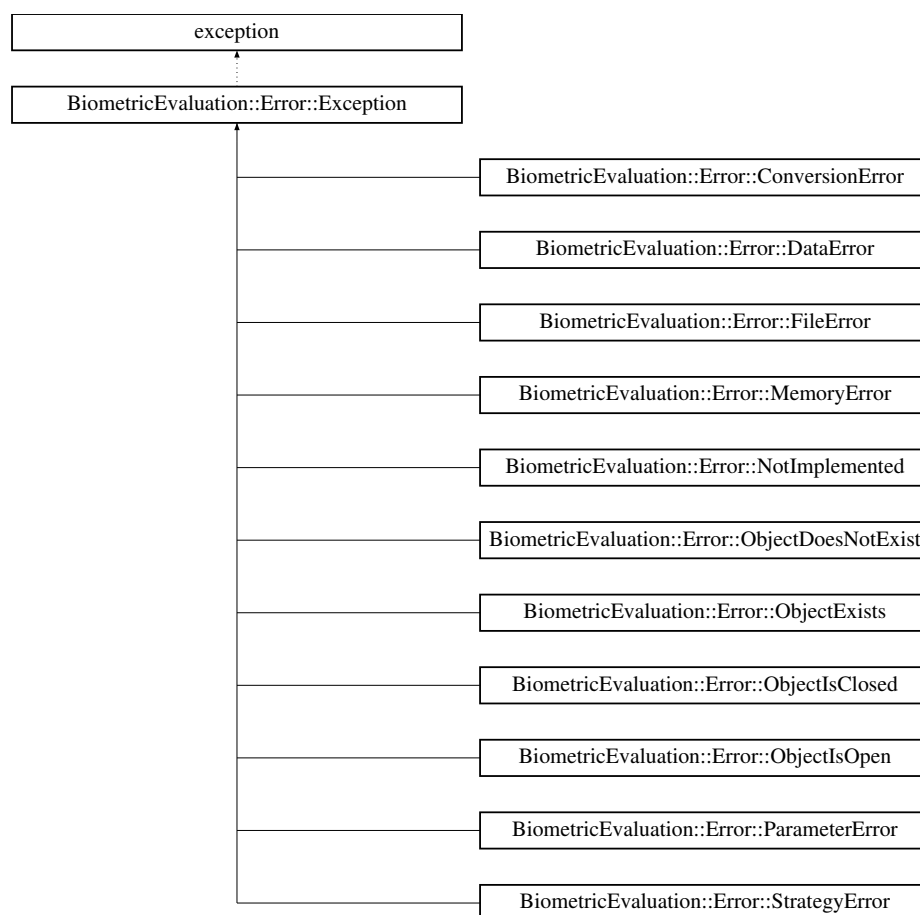
Implicit conversion to enumeration

## F.37 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](#) (std::string info)
- const char \* [what](#) () const noexcept
- const std::string [whatString](#) () const noexcept

#### F.37.1 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.

### F.37.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::Exception::Exception ( )**

Construct an [Exception](#) object without an information string.

**BiometricEvaluation::Error::Exception::Exception ( std::string *info* )**

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

Parameters

<i>in</i>	<i>info</i>	The information string associated with the exception.
-----------	-------------	---

### F.37.3 Member Function Documentation

**const char\* BiometricEvaluation::Error::Exception::what ( ) const [noexcept]**

Obtain the information string associated with the exception.

Returns

The information string as a char array.

**const std::string BiometricEvaluation::Error::Exception::whatString ( ) const [noexcept]**

Obtain the information string associated with the exception.

Returns

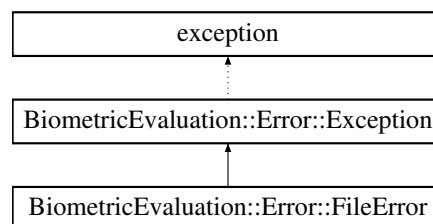
The information string.

## F.38 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](#) (std::string info)

### F.38.1 Detailed Description

File error when opening, reading, writing, etc.

### F.38.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::FileError::FileError ( )**

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

**BiometricEvaluation::Error::FileError::FileError ( std::string *info* )**

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

## F.39 BiometricEvaluation::IO::FileLogCabinet Class Reference

```
#include <be_io_filelogcabinet.h>
```

### Public Member Functions

- [FileLogCabinet](#) (const std::string &pathname, const std::string &description)
- [FileLogCabinet](#) (const std::string &pathname)
- std::shared\_ptr< [FileLogsheet](#) > [newLogsheet](#) (const std::string &name, const std::string &description)
- std::string [getPathname](#) ()
- std::string [getDescription](#) ()
- unsigned int [getCount](#) ()

### F.39.1 Detailed Description

A class to represent a collection of log sheets.

### F.39.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::FileLogCabinet::FileLogCabinet ( const std::string & *pathname*, const std::string & *description* )**

Create a new [FileLogCabinet](#) in the file system.

Parameters

in	<i>pathname</i>	The pathname where the <a href="#">FileLogCabinet</a> is to be created.
in	<i>description</i>	The text used to describe the cabinet.

Exceptions

<a href="#">Error::ObjectExists</a>	The cabinet was previously created.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying file system.

**BiometricEvaluation::IO::FileLogCabinet::FileLogCabinet ( const std::string & *pathname* )**

Open an existing [FileLogCabinet](#).

## Parameters

in	<i>pathname</i>	The pathname where the <a href="#">FileLogCabinet</a> is located.
----	-----------------	---

## Exceptions

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

### F.39.3 Member Function Documentation

#### **unsigned int BiometricEvaluation::IO::FileLogCabinet::getCount ( )**

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

@ returns The number of logsheets manages by the cabinet.

#### **std::string BiometricEvaluation::IO::FileLogCabinet::getDescription ( )**

Obtain the description of the [FileLogCabinet](#).

@ returns The description of the [FileLogCabinet](#).

#### **std::string BiometricEvaluation::IO::FileLogCabinet::getPathname ( )**

Obtain the pathname of the [FileLogCabinet](#).

@ returns The pathname of the [FileLogCabinet](#).

#### **std::shared\_ptr<FileLogsheet> BiometricEvaluation::IO::FileLogCabinet::newLogsheet ( const std::string & name, const std::string & description )**

Create a new [FileLogsheet](#) within the cabinet.

## Parameters

in	<i>name</i>	The name of the <a href="#">FileLogsheet</a> to be created. This can not be a path name.
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

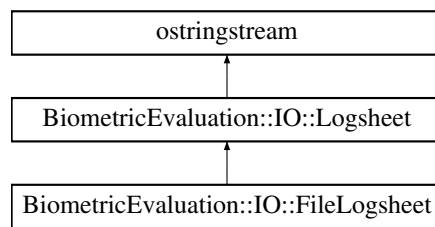
<a href="#"><i>Error::ObjectExists</i></a>	The sheet was previously created.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when using the underlying file system.

## F.40 BiometricEvaluation::IO::FileLogsheet Class Reference

A class to represent a single logging mechanism with a file as the backing store.

```
#include <be_io_filelogsheet.h>
```

Inheritance diagram for BiometricEvaluation::IO::FileLogsheet:



## Public Member Functions

- [FileLogsheets](#) (const std::string &url, const std::string &description)  
*Create a new log sheet.*
- [FileLogsheets](#) (const std::string &url)  
*Open an existing log sheet for appending.*
- [~FileLogsheets](#) ()
- std::string [sequence](#) (bool allEntries=false, bool [trim](#)=true, int32\_t cursor=[BE\\_FILELOGSHEET\\_SEQ←\\_NEXT](#))  
*Sequence through a [FileLogsheets](#), returning one entry per invocation.*
- void [write](#) (const std::string &entry)  
*Write a string as an entry to the backing store.*
- void [writeComment](#) (const std::string &entry)  
*Write a string as a comment to the backing store.*
- void [writeDebug](#) (const std::string &entry)  
*Write a string as a debug entry to the backing store.*
- void [sync](#) ()  
*Synchronize any buffered data to the underlying backing store.*

## Static Public Member Functions

- static void [mergeLogsheets](#) (std::vector< std::shared\_ptr< [FileLogsheets](#) >> &logsheets)  
*Merge multiple [FileLogsheets](#) into a single [FileLogsheets](#).*
- static std::string [trim](#) (const std::string &entry)  
*Trim delimiters from [FileLogsheets](#) entries.*

## Static Public Attributes

- static const int32\_t [BE\\_FILELOGSHEET\\_SEQ\\_START](#) = 1
- static const int32\_t [BE\\_FILELOGSHEET\\_SEQ\\_NEXT](#) = 2

## Protected Member Functions

- [FileLogsheets](#) (const [FileLogsheets](#) &)
- [FileLogsheets](#) & [operator=](#) (const [FileLogsheets](#) &)
- void [updateCursor](#) ()  
*Update the cursor position of the sequence file.*



## Protected Attributes

- `std::auto_ptr< std::fstream > _theLogFile`
- `std::shared_ptr< std::fstream > _sequenceFile`
- `streamoff _cursor`

## Additional Inherited Members

### F.40.1 Detailed Description

A class to represent a single logging mechanism with a file as the backing store.

A [FileLogsheet](#) object can be constructed and passed back to the client by the LogCabinet object. All sheets created in this manner are placed in a common area maintained by the cabinet.

### F.40.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::FileLogsheet::FileLogsheet ( const std::string & *url*, const std::string & *description* )**

Create a new log sheet.

the log sheet is named by the uniform resource locator, usually starting with '[file:///](#)'. However, relative and absolute path names are also accepted for backward compatibility.

Parameters

<code>in</code>	<code>url</code>	The Uniform Resource Locator of the <a href="#">FileLogsheet</a> to be created.
<code>in</code>	<code>description</code>	The text used to describe the sheet. This text is written into the log file prior to any entries.

Exceptions

<a href="#">Error::ParameterError</a>	The URL is malformed.
<a href="#">Error::ObjectExists</a>	The sheet was previously created.
<a href="#">Error::StrategyError</a>	An error occurred when using the underlying file system, or name or parentDir is malformed.

**BiometricEvaluation::IO::FileLogsheet::FileLogsheet ( const std::string & *url* )**

Open an existing log sheet for appending.

On open, the current entry counter is set to the last entry number plus one.

Note

Opening a large [FileLogsheet](#) may be a costly operation.

Parameters

<code>in</code>	<code>url</code>	The Uniform Resource Locator of the <a href="#">FileLogsheet</a> to be opened.
-----------------	------------------	--

Exceptions

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

**BiometricEvaluation::IO::FileLogsheet::~~FileLogsheet ( )**

Destructor

**BiometricEvaluation::IO::FileLogsheet::FileLogsheet ( const FileLogsheet & ) [protected]**

Prevent copying of [FileLogsheet](#) objects

### F.40.3 Member Function Documentation

**static void BiometricEvaluation::IO::FileLogsheet::mergeLogsheets ( std::vector< std::shared\_ptr< FileLogsheet >> & logsheets ) [static]**

Merge multiple FileLogsheets into a single [FileLogsheet](#).

[Logsheet](#) 2 - n will be appended to [Logsheet](#) 1.

Parameters

<i>logSheets</i>	<a href="#">Logsheet</a> to merge.
------------------	------------------------------------

Exceptions

<i>Error::FileError</i>	Error during log sequence.
<i>Error::StrategyError</i>	Error during log sequence.

**FileLogsheet& BiometricEvaluation::IO::FileLogsheet::operator= ( const FileLogsheet & ) [protected]**

Prevent copying of [FileLogsheet](#) objects

**std::string BiometricEvaluation::IO::FileLogsheet::sequence ( bool allEntries = false, bool trim = true, int32\_t cursor = BE\_FILELOGSHEET\_SEQ\_NEXT )**

Sequence through a [FileLogsheet](#), returning one entry per invocation.

Parameters

<i>allEntries</i>	Include debug and comment entries 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

<i>Error::FileError, Error</i>	occured while performing file <a href="#">IO</a> .
<i>Error::ObjectDoesNotExist</i>	The <a href="#">FileLogsheet</a> cannot be found on disk.
<i>Error::StrategyError</i>	Invalid cursor position or the contents of the <a href="#">FileLogsheet</a> is malformed.

**void BiometricEvaluation::IO::FileLogsheet::sync ( ) [virtual]**

Synchronize any buffered data to the underlying backing store.

This syncing is dependent on the behavior of the underlying storage mechanism.

Exceptions

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

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

**static std::string BiometricEvaluation::IO::FileLogsheet::trim ( const std::string & entry )  
[static]**

Trim delimiters from [FileLogsheet](#) entries.

Works for comments and numbered entries.

Parameters

<i>in</i>	<i>entry</i>	The entry to trim.
-----------	--------------	--------------------

Returns

Delimiter-less entry.

**void BiometricEvaluation::IO::FileLogsheet::updateCursor ( ) [protected]**

Update the cursor position of the sequence file.

Exceptions

<a href="#"><i>Error::FileError</i></a>	<a href="#">Error</a> getting file position from sequence file.
---	---

**void BiometricEvaluation::IO::FileLogsheet::write ( const std::string & entry ) [virtual]**

Write a string as an entry to the backing store.

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

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

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

**void BiometricEvaluation::IO::FileLogsheet::writeComment ( const std::string & entry )  
[virtual]**

Write a string as a comment to the backing store.

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>entry</i>	The text of the comment.
-----------	--------------	--------------------------

Exceptions

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

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

**void BiometricEvaluation::IO::FileLogsheet::writeDebug ( const std::string & *entry* ) [virtual]**

Write a string as a debug entry to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A debug line is prefixed with DebugDelimiter followed by a space.

Parameters

<i>in</i>	<i>entry</i>	The text of the debug message.
-----------	--------------	--------------------------------

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when logging.
---	---------------------------------

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

## F.40.4 Member Data Documentation

**streamoff BiometricEvaluation::IO::FileLogsheet::\_cursor [protected]**

Position of the sequencer, relative to SOF

**std::shared\_ptr<std::fstream> BiometricEvaluation::IO::FileLogsheet::\_sequenceFile [protected]**

Stream used for sequencing

**std::auto\_ptr<std::fstream> BiometricEvaluation::IO::FileLogsheet::\_theLogFile [protected]**

Stream used for writing the log file

**const int32\_t BiometricEvaluation::IO::FileLogsheet::BE\_FILELOGSHEET\_SEQ\_NEXT = 2 [static]**

Sequence from current position

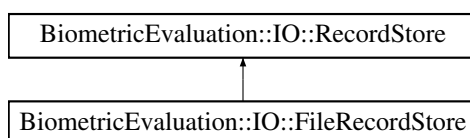
**const int32\_t BiometricEvaluation::IO::FileLogsheet::BE\_FILELOGSHEET\_SEQ\_START = 1 [static]**

Sequence from beginning

## F.41 BiometricEvaluation::IO::FileRecordStore Class Reference

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

Inheritance diagram for BiometricEvaluation::IO::FileRecordStore:



## Public Member Functions

- [FileRecordStore](#) (const std::string &pathname, const std::string &description)
- [FileRecordStore](#) (const std::string &name, uint8\_t mode=IO::READWRITE)
- uint64\_t [getSpaceUsed](#) () const  
*Obtain real storage utilization.*
- void [insert](#) (const std::string &key, const void \*const data, const uint64\_t size)
- void [remove](#) (const std::string &key)
- uint64\_t [read](#) (const std::string &key, void \*const data) const
- virtual void [replace](#) (const std::string &key, const void \*const data, const uint64\_t size)
- virtual uint64\_t [length](#) (const std::string &key) const
- void [flush](#) (const std::string &key) const
- uint64\_t [sequence](#) (std::string &key, void \*const data=nullptr, int cursor=BE\_RECSTORE\_SEQ\_NEXT)  
*Sequence through a [RecordStore](#), returning the key/data pairs.*
- void [setCursorAtKey](#) (const std::string &key)
- void [move](#) (const std::string &pathname)  
*Move the [RecordStore](#).*
- **FileRecordStore** (const [FileRecordStore](#) &)=delete
- [FileRecordStore](#) & **operator=** (const [FileRecordStore](#) &)=delete

## Protected Member Functions

- std::string [canonicalName](#) (const std::string &name) const

## Additional Inherited Members

### F.41.1 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.

### F.41.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::FileRecordStore::FileRecordStore** ( const std::string & *pathname*, const std::string & *description* )

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

Parameters

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

Exceptions

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

---

**BiometricEvaluation::IO::FileRecordStore::FileRecordStore** ( const std::string & *name*, uint8\_t *mode* = *IO::READWRITE* )

Open an existing [FileRecordStore](#).

## Parameters

in	<i>name</i>	The path name of the store.
in	<i>mode</i>	Open mode, read-only or read-write.

## Exceptions

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

**F.41.3 Member Function Documentation**

**void BiometricEvaluation::IO::FileRecordStore::flush ( const std::string & *key* ) const [virtual]**

Commit the record's data to storage.

## Parameters

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

## Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::FileRecordStore::getSpaceUsed ( ) const [virtual]**

Obtain real storage utilization.

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

## Returns

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

## Exceptions

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

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

**void BiometricEvaluation::IO::FileRecordStore::insert ( const std::string & *key*, const void \*const *data*, const uint64\_t *size* ) [virtual]**

Insert a record into the store.

## Parameters

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

Exceptions

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

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

**virtual uint64\_t BiometricEvaluation::IO::FileRecordStore::length ( const std::string & *key* ) const**  
[**virtual**]

Return the length of a record.

Parameters

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

Returns

The record length.

Exceptions

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

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

**void BiometricEvaluation::IO::FileRecordStore::move ( const std::string & *pathname* )** [**virtual**]

Move the [RecordStore](#).

The [RecordStore](#) can be moved to a new path in the file system.

Parameters

in	<i>pathname</i>	The new path of the <a href="#">RecordStore</a> .
----	-----------------	---

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::FileRecordStore::read ( const std::string & *key*, void \*const *data* )**  
**const** [**virtual**]

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

Parameters

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

Returns

The size of the record.



Exceptions

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

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

**void BiometricEvaluation::IO::FileRecordStore::remove ( const std::string & key ) [virtual]**

Remove a record from the store.

Parameters

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

Exceptions

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

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

**virtual void BiometricEvaluation::IO::FileRecordStore::replace ( const std::string & key, const void \*const data, const uint64\_t size ) [virtual]**

Replace a complete record in a store.

Parameters

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

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::FileRecordStore::sequence ( std::string & key, void \*const data = nullptr, int cursor = BE\_RECSTORE\_SEQ\_NEXT ) [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 nullptr to indicate only the key is wanted.

**F.42 BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem Struct Reference 233**

<code>in</code>	<code>cursor</code>	The location within the sequence of the key/data pair to return.
-----------------	---------------------	--

Returns

The length of the record currently in sequence.

Exceptions

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

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

**void BiometricEvaluation::IO::FileRecordStore::setCursorAtKey ( const std::string & *key* )**  
**[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

<code>in</code>	<code>key</code>	The key of the record which will be returned by the first subsequent call to <a href="#">sequence()</a> .
-----------------	------------------	---

Exceptions

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

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

**F.42 BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem Struct Reference**

Representation of information about a fingerprint reader system.

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

Public Attributes

- std::string [name](#)
- [EncodingMethod](#) [method](#)
- std::string [equipment](#)

**F.42.1 Detailed Description**

Representation of information about a fingerprint reader system.

**F.42.2 Member Data Documentation**

**std::string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::equipment**

Optional ID for equipment used in system

**EncodingMethod BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem**↵  
**::method**

Method used to encoded minutiae

**std::string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::name**

Name for system that encoded minutiae

## F.43 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](#) *fingerPosition*, const [Image::CoordinateSet](#) *coordinates*)  
 Create an [FingerSegmentPosition](#) struct.

### Public Attributes

- [Finger::Position](#) *fingerPosition*
- [Image::CoordinateSet](#) *coordinates*

#### F.43.1 Detailed Description

Locations of an individual finger segment in a slap.

#### F.43.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::FingerSegmentPosition** (  
 const [Finger::Position](#) *fingerPosition*, const [Image::CoordinateSet](#) *coordinates* )

Create an [FingerSegmentPosition](#) struct.

Parameters

<i>fingerPosition</i>	<a href="#">Finger</a> depicted in this segment.
<i>coordinates</i>	Collection of coordinates that compose the segment bonding polygon.

#### F.43.3 Member Data Documentation

**Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition**↵  
**::coordinates**

Points composing the segmented polygon

**Finger::Position BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::finger**↵  
**Position**

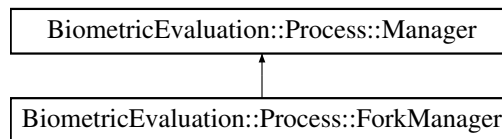
[Finger](#) depicted in this segment

## F.44 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** ()
- std::shared\_ptr< **WorkerController** > **addWorker** (std::shared\_ptr< **Worker** > worker)  
*Adds a **Worker** to be managed by this **Manager**.*
- void **startWorkers** (bool wait=true, bool communicate=false)  
*Begin **Worker**'s work.*
- void **startWorker** (std::shared\_ptr< **WorkerController** > worker, bool wait=true, bool communicate=false)  
*Start a worker.*
- int32\_t **stopWorker** (std::shared\_ptr< **WorkerController** > workerController)  
*Ask **Worker** to exit.*
- void **broadcastSignal** (int signo)  
*Send a POSIX signal to all workers.*
- bool **responsibleFor** (const pid\_t pid) const  
*Obtain whether or not this **ForkManager** is responsible for a particular PID.*
- void **setNotWorking** (const pid\_t pid)  
*Set Status.isWorking for PID to false.*
- void **markAllFinished** ()  
*Call **setNotWorking()** for all PIDs known to this **ForkManager**.*
- bool **getIsWorkingStatus** (const pid\_t pid) const  
*Get Status.isWorking for PID.*
- void **waitForWorkerExit** ()  
*Block until all Workers have exited.*
- **~ForkManager** ()  
***ForkManager** destructor.*
- void **setExitCallback** (void(\*exitCallback)(std::shared\_ptr< **ForkWorkerController** > worker, int status, ← loc))  
*Call a function in your program when a child exits.*

### Static Public Member Functions

- static void **defaultExitCallback** (std::shared\_ptr< **ForkWorkerController** > worker, int status)  
*A default exit callback function.*

## Static Public Attributes

- static std::list< [ForkManager](#) \* > [FORKMANAGERS](#)

*List of all instantiated ForkManagers.*

## Additional Inherited Members

### F.44.1 Detailed Description

[Manager](#) implementation that starts Workers by calling fork(2).

### F.44.2 Constructor & Destructor Documentation

**BiometricEvaluation::Process::ForkManager::ForkManager ( )**

[ForkManager](#) constructor.

### F.44.3 Member Function Documentation

**std::shared\_ptr<WorkerController> BiometricEvaluation::Process::ForkManager::addWorker ( std::shared\_ptr< Worker > worker ) [virtual]**

Adds a [Worker](#) to be managed by this [Manager](#).

Parameters

<i>worker</i>	A <a href="#">Worker</a> instance to run.
---------------	---

Returns

shared\_ptr to worker.

Implements [BiometricEvaluation::Process::Manager](#).

**void BiometricEvaluation::Process::ForkManager::broadcastSignal ( int signo )**

Send a POSIX signal to all workers.

Parameters

<i>in</i>	<i>signo</i>	The signal to send.
-----------	--------------	---------------------

**static void BiometricEvaluation::Process::ForkManager::defaultExitCallback ( std::shared\_ptr< ForkWorkerController > worker, int status ) [static]**

A default exit callback function.

Writes to stdout in the form: PID #: Exited .

Parameters

<i>worker</i>	The <a href="#">ForkWorkerController</a> object that exited.
<i>status</i>	The status of the <a href="#">Worker</a> that exited (from wait(2)).

**bool BiometricEvaluation::Process::ForkManager::getIsWorkingStatus ( const pid\_t pid ) const**

Get Status.isWorking for PID.

## Parameters

<code>in</code>	<code>pid</code>	PID whose inWorking flag should be queried
-----------------	------------------	--

## Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	PID not under this manager's control.
--	---------------------------------------

**bool BiometricEvaluation::Process::ForkManager::responsibleFor ( const pid\_t *pid* ) const**

Obtain whether or not this [ForkManager](#) is responsible for a particular PID.

## Parameters

<code>in</code>	<code>pid</code>	PID in question
-----------------	------------------	-----------------

## Returns

true if this [ForkManager](#) spawned pid, false otherwise.

**void BiometricEvaluation::Process::ForkManager::setExitCallback ( void(\*)(std::shared\_ptr< ForkWorkerController > worker, int stat\_loc) *exitCallback* )**

Call a function in your program when a child exits.

## Parameters

<code><i>exitCallback</i></code>	Function pointer to a method that takes a shared_ptr to a <a href="#">ForkWorkerController</a> and the integer status information.
----------------------------------	--

## Note

The exit callback will not have any effect if the [Manager](#) is not set to wait for Workers.

**void BiometricEvaluation::Process::ForkManager::setNotWorking ( const pid\_t *pid* )**

Set Status.isWorking for PID to false.

## Parameters

<code>in</code>	<code>pid</code>	PID whose inWorking flag should be set to false
-----------------	------------------	---

## Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	PID not under this manager's control.
--	---------------------------------------

**void BiometricEvaluation::Process::ForkManager::startWorker ( std::shared\_ptr< WorkerController > *worker*, bool *wait* = *true*, bool *communicate* = *false* ) [virtual]**

Start a worker.

## Parameters

	<i>worker</i>	Pointer to a <a href="#">WorkerController</a> that is being managed by this <a href="#">Manager</a> instance.
	<i>wait</i>	Whether or not to wait for this <a href="#">Worker</a> to exit before returning control to the caller.
<i>in</i>	<i>communicate</i>	Whether or not to enable communication among the Workers and Managers.

## Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	worker is already working.
<a href="#"><i>Error::StrategyError</i></a>	worker is not managed by this <a href="#">Manager</a> instance.

Implements [BiometricEvaluation::Process::Manager](#).

**void BiometricEvaluation::Process::ForkManager::startWorkers ( bool *wait* = *true*, bool *communicate* = *false* ) [virtual]**

Begin [Worker](#)'s work.

## Parameters

<i>in</i>	<i>wait</i>	Whether or not to wait for all Workers to return before returning.
<i>in</i>	<i>communicate</i>	Whether or not to enable communication among the Workers and Managers.

## Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	At least one <a href="#">Worker</a> is already working.
<a href="#"><i>Error::StrategyError</i></a>	Problem forking.

Implements [BiometricEvaluation::Process::Manager](#).

**int32\_t BiometricEvaluation::Process::ForkManager::stopWorker ( std::shared\_ptr< [WorkerController](#) > *workerController* ) [virtual]**

Ask [Worker](#) to exit.

Sends SIGUSR1 to the [Worker](#), which [ForkManager](#) will handle automatically.

## Parameters

<i>worker↵ Controller</i>	Pointer to the <a href="#">ForkWorkerController</a> that should be stopped.
-------------------------------	---

## Returns

Exit status of worker.

## Exceptions

<a href="#"><i>Error::ObjectDoesNot↵ Exist</i></a>	worker is not working.
<a href="#"><i>Error::StrategyError</i></a>	Problem sending the signal.

## Attention

Do not call [stopWorker\(\)](#) when communication is enabled unless you will be finished with communication for all Workers at that point. This creates a race condition for reads()/writes() when the [Worker](#) exits.

Implements [BiometricEvaluation::Process::Manager](#).

**void BiometricEvaluation::Process::ForkManager::waitForWorkerExit ( ) [virtual]**

Block until all Workers have exited.

Use this method if wait=false was set during a call to startWorker(s) but now wait=true is desired.

Implements [BiometricEvaluation::Process::Manager](#).

#### F.44.4 Member Data Documentation

**std::list<ForkManager\*> BiometricEvaluation::Process::ForkManager::FORKMANAGERS [static]**

List of all instantiated ForkManagers.

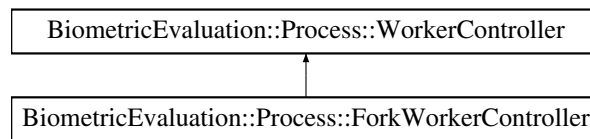
This is not a list of managed pointers to ForkManagers. If it was, the smart pointer's destructor would attempt to delete the object being pointed to at program termination, which is ultimately sometime after the destructor of the [ForkManager](#) itself was called.

### F.45 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.*
- bool [everWorked](#) () const  
*Obtain whether or not this [Worker](#) has ever worked.*
- void [reset](#) ()  
*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 [\\_staticWorker](#) to stop.*



## Friends

- void [ForkManager::startWorkers](#) (bool wait, bool communicate)  
*Begin [Worker](#)'s work.*
- void [ForkManager::startWorker](#) (std::shared\_ptr< [WorkerController](#) > worker, bool wait, bool communicate)  
*Restart a completed [Worker](#).*
- int32\_t [ForkManager::stopWorker](#) (std::shared\_ptr< [WorkerController](#) > workerController)  
*Ask [Worker](#) to exit.*
- std::shared\_ptr< [WorkerController](#) > [ForkManager::addWorker](#) (std::shared\_ptr< [Worker](#) > worker)  
*Adds a [Worker](#) to be managed by this [Manager](#).*

## Additional Inherited Members

### F.45.1 Detailed Description

Wrapper of a [Worker](#) returned from a [Process::ForkManager](#).

### F.45.2 Member Function Documentation

**static void BiometricEvaluation::Process::ForkWorkerController::stop ( int *signal* ) [static]**

Tell `_staticWorker` 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).
---------------	---

**bool BiometricEvaluation::Process::ForkWorkerController::everWorked ( ) const [virtual]**

Obtain whether or not this [Worker](#) has ever worked.

Returns

true the [Worker](#) has ever or is currently working, false otherwise.

Note

[reset\(\)](#) will change the result of this method.

Implements [BiometricEvaluation::Process::WorkerController](#).

**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.

**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](#).

**void BiometricEvaluation::Process::ForkWorkerController::reset ( ) [virtual]**

Reuse the [Worker](#).

Exceptions

<a href="#">Error::ObjectExists</a>	The previously started <a href="#">Worker</a> is still running.
-------------------------------------	---

Reimplemented from [BiometricEvaluation::Process::WorkerController](#).

### F.45.3 Friends And Related Function Documentation

**std::shared\_ptr<WorkerController> ForkManager::addWorker ( std::shared\_ptr< Worker > *worker* ) [friend]**

Adds a [Worker](#) to be managed by this [Manager](#).

Parameters

<i>worker</i>	A <a href="#">Worker</a> instance to run.
---------------	---

Returns

shared\_ptr to worker.

**void ForkManager::startWorker ( std::shared\_ptr< WorkerController > *worker*, bool *wait*, bool *communicate* ) [friend]**

Restart a completed [Worker](#).

Parameters

	<i>worker</i>	Pointer to a <a href="#">WorkerController</a> that is being managed by this <a href="#">Manager</a> instance.
	<i>wait</i>	Whether or not to wait for this <a href="#">Worker</a> to exit before returning control to the caller.
<i>in</i>	<i>communicate</i>	Whether or not to enable communication among the Workers and Managers.

Exceptions

<a href="#">Error::ObjectExists</a>	worker is already working.
<a href="#">Error::StrategyError</a>	worker is not managed by this <a href="#">Manager</a> instance.

**void ForkManager::startWorkers ( bool *wait*, bool *communicate* ) [friend]**

Begin [Worker](#)'s work.

## Parameters

in	<i>wait</i>	Whether or not to wait for all Workers to return before returning.
in	<i>communicate</i>	Whether or not to enable communication among the Workers and Managers.

## Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	One or more of the Workers is already working.
<a href="#"><i>Error::StrategyError</i></a>	Problem forking.

**int32\_t ForkManager::stopWorker ( std::shared\_ptr< WorkerController > workerController )**  
**[friend]**

Ask [Worker](#) to exit.

Sends SIGUSR1 to the [Worker](#), which [ForkManager](#) will handle automatically.

## Parameters

<i>worker↵ Controller</i>	Pointer to the <a href="#">ForkWorkerController</a> that should be stopped.
-------------------------------	---

## Returns

Exit status of worker.

## Exceptions

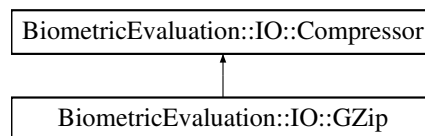
<a href="#"><i>Error::ObjectDoesNot↵ Exist</i></a>	worker is not working.
<a href="#"><i>Error::StrategyError</i></a>	Problem sending the signal.

## F.46 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 uncompressedData↵Size) const  
*Compress a buffer.*
- [Memory::uint8Array compress](#) (const [Memory::uint8Array](#) &uncompressedData) const  
*Compress a buffer.*
- void [compress](#) (const uint8\_t \*const uncompressedData, uint64\_t uncompressedDataSize, const std↵::string &outputFile) const

- Compress a buffer.*
- void [compress](#) (const [Memory::uint8Array](#) &uncompressedData, const std::string &outputFile) const
- Compress a buffer.*
- [Memory::uint8Array compress](#) (const std::string &inputFile) const
- Compress a file.*
- void [compress](#) (const std::string &inputFile, const std::string &outputFile) const
- Compress a file.*
- [Memory::uint8Array decompress](#) (const uint8\_t \*const compressedData, uint64\_t compressedDataSize) const
- Decompress a compressed buffer.*
- [Memory::uint8Array decompress](#) (const [Memory::uint8Array](#) &compressedData) const
- Decompress a compressed buffer.*
- [Memory::uint8Array decompress](#) (const std::string &input) const
- Decompress a compressed buffer into a file.*
- void [decompress](#) (const std::string &inputFile, const std::string &outputFile) const
- Decompress a file.*
- void [decompress](#) (const uint8\_t \*const compressedData, const uint64\_t compressedDataSize, const std::string &outputFile) const
- Decompress a file.*
- void [decompress](#) (const [Memory::uint8Array](#) &compressedData, const std::string &outputFile) const
- Decompress a file.*
- [GZip](#) (const [GZip](#) &other)=delete
- Copy constructor (disabled).*
- [GZip & operator=](#) (const [GZip](#) &other)=delete
- Assignment overload (disabled).*

## Static Public Attributes

- static const std::string [COMPRESSION\\_LEVEL](#)
- static const std::string [COMPRESSION\\_STRATEGY](#)
- static const std::string [COMPRESSION\\_METHOD](#)
- static const std::string [INPUT\\_DATA\\_TYPE](#)
- static const std::string [WINDOW\\_BITS](#)
- static const std::string [MEMORY\\_LEVEL](#)
- static const std::string [CHUNK\\_SIZE](#)

## Additional Inherited Members

### F.46.1 Detailed Description

[Compressor](#) for gzip compression from zlib.

### F.46.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::GZip::GZip ( const [GZip](#) & other ) [delete]**

Copy constructor (disabled).

Disabled because [Properties](#) member of parent cannot be copied.

Parameters

<i>other</i>	<a href="#">GZip</a> to copy.
--------------	-------------------------------

### F.46.3 Member Function Documentation

**Memory::uint8Array BiometricEvaluation::IO::GZip::compress ( const uint8\_t \*const *uncompressedData*, uint64\_t *uncompressedDataSize* ) const [virtual]**

Compress a buffer.

Parameters

<i>uncompressedData</i>	Uncompressed data buffer to compress.
<i>uncompressedDataSize</i>	Size of <i>uncompressedData</i> .

Returns

Compressed buffer.

Exceptions

<a href="#">Error::StrategyError</a>	Error in compression unit.
--------------------------------------	----------------------------

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

**Memory::uint8Array BiometricEvaluation::IO::GZip::compress ( const Memory::uint8Array & *uncompressedData* ) const [virtual]**

Compress a buffer.

Parameters

<i>uncompressedData</i>	Uncompressed data buffer to compress.
-------------------------	---------------------------------------

Returns

Compressed buffer.

Exceptions

<a href="#">Error::StrategyError</a>	Error in decompression unit.
--------------------------------------	------------------------------

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

**void BiometricEvaluation::IO::GZip::compress ( const uint8\_t \*const *uncompressedData*, uint64\_t *uncompressedDataSize*, const std::string & *outputFile* ) const [virtual]**

Compress a buffer.

Parameters

<i>uncompressedData</i>	Uncompressed data buffer to compress.
-------------------------	---------------------------------------

<i>uncompressedDataSize</i>	Size of uncompressedData.
<i>outputFile</i>	Location to save compressed file.

Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in compression unit.

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

**void BiometricEvaluation::IO::GZip::compress ( const Memory::uint8Array & *uncompressedData*, const std::string & *outputFile* ) const [virtual]**

Compress a buffer.

Parameters

<i>uncompressedData</i>	Uncompressed data buffer to compress.
<i>outputFile</i>	Location to save compressed file.

Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in decompression unit.

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

**Memory::uint8Array BiometricEvaluation::IO::GZip::compress ( const std::string & *inputFile* ) const [virtual]**

Compress a file.

Parameters

<i>inputFile</i>	Path to file to compress.
------------------	---------------------------

Returns

Compressed buffer.

Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	Input file does not exist.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in decompression unit.

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

**void BiometricEvaluation::IO::GZip::compress ( const std::string & *inputFile*, const std::string & *outputFile* ) const [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

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	Input file does not exist.
<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in decompression unit.

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

**Memory::uint8Array BiometricEvaluation::IO::GZip::decompress ( const uint8\_t \*const compressedData, uint64\_t compressedDataSize ) const [virtual]**

Decompress a compressed buffer.

Parameters

<i>compressedData</i>	Compressed data buffer to decompress.
<i>compressedDataSize</i>	Size of compressedData.

Returns

Decompressed data.

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in compression unit.
---	--

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

**Memory::uint8Array BiometricEvaluation::IO::GZip::decompress ( const Memory::uint8Array & compressedData ) const [virtual]**

Decompress a compressed buffer.

Parameters

<i>compressedData</i>	Compressed data buffer to decompress.
-----------------------	---------------------------------------

Returns

Decompressed data.

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> in decompression unit.
---	--

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

**Memory::uint8Array BiometricEvaluation::IO::GZip::decompress ( const std::string & inputFile ) const [virtual]**

Decompress a compressed buffer into a file.

## Parameters

<i>inputFile</i>	Location to save compressed file.
------------------	-----------------------------------

## Returns

Decompressed data.

## Exceptions

<a href="#"><i>Error::StrategyError</i></a>	Error in decompression unit.
<a href="#"><i>Error::ObjectDoesNotExist</i></a>	Output file already exists.

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

**void BiometricEvaluation::IO::GZip::decompress ( const std::string & *inputFile*, const std::string & *outputFile* ) const** **[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

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	Input file does not exist.
<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	Error in compression unit.

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

**void BiometricEvaluation::IO::GZip::decompress ( const uint8\_t \*const *compressedData*, const uint64\_t *compressedDataSize*, const std::string & *outputFile* ) const** **[virtual]**

Decompress a file.

## Parameters

<i>compressedData</i>	Compressed data buffer to decompress.
<i>compressedDataSize</i>	Size of compressedData.
<i>outputFile</i>	Path to location where decompressed version will be saved.

## Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	Error in compression unit.

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

**void BiometricEvaluation::IO::GZip::decompress ( const Memory::uint8Array & *compressedData*, const std::string & *outputFile* ) const** **[virtual]**

Decompress a file.



Parameters

<i>compressedData</i>	Compressed data buffer to decompress.
<i>outputFile</i>	Path to location where decompressed version will be saved.

Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	Output file already exists.
<a href="#"><i>Error::StrategyError</i></a>	Error in compression unit.

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

**GZip& BiometricEvaluation::IO::GZip::operator= ( const GZip & other ) [delete]**

Assignment overload (disabled).

Disabled because [Properties](#) member of parent cannot be assigned.

Parameters

<i>other</i>	<a href="#">GZip</a> to assign.
--------------	---------------------------------

Returns

lhs [GZip](#).

## F.46.4 Member Data Documentation

**const std::string BiometricEvaluation::IO::GZip::CHUNK\_SIZE [static]**

How many bytes to work at a time

**const std::string BiometricEvaluation::IO::GZip::COMPRESSION\_LEVEL [static]**

How thorough the compression should be

**const std::string BiometricEvaluation::IO::GZip::COMPRESSION\_METHOD [static]**

Which underlying method in the compressor

**const std::string BiometricEvaluation::IO::GZip::COMPRESSION\_STRATEGY [static]**

Which underlying algorithm to use

**const std::string BiometricEvaluation::IO::GZip::INPUT\_DATA\_TYPE [static]**

The type of data being compressed

**const std::string BiometricEvaluation::IO::GZip::MEMORY\_LEVEL [static]**

How much memory for internal compression state

**const std::string BiometricEvaluation::IO::GZip::WINDOW\_BITS [static]**

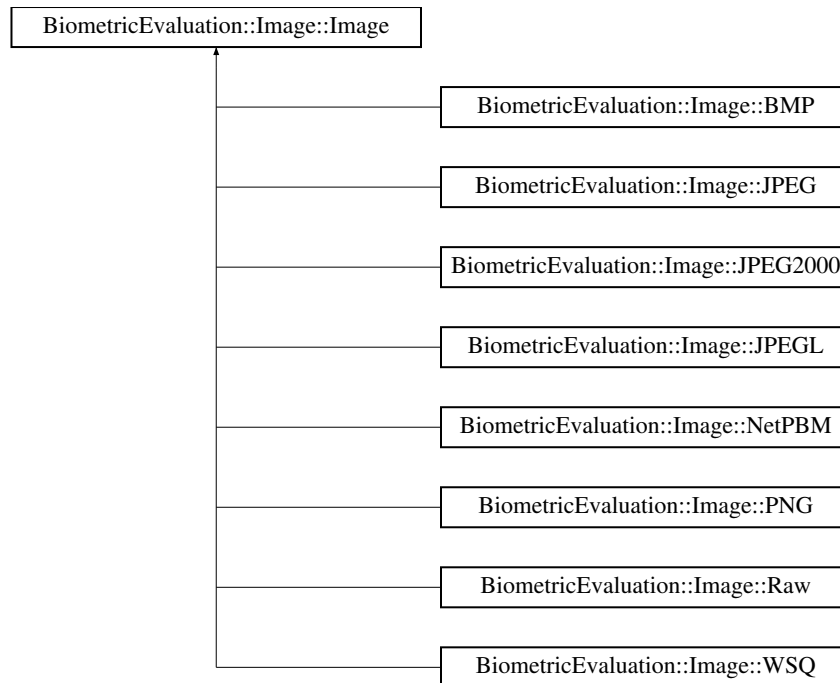
Window size

## F.47 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](#) compression)  
*Parent constructor for all [Image](#) classes.*
- **Image** (const uint8\_t \*data, const uint64\_t size, const [CompressionAlgorithm](#) compression)  
*Parent constructor for all [Image](#) classes.*
- **CompressionAlgorithm** [getCompressionAlgorithm](#) () const  
*Accessor for the [CompressionAlgorithm](#) of the image.*
- **Resolution** [getResolution](#) () const  
*Accessor for the resolution of the image.*
- **Memory::uint8Array** [getData](#) () const  
*Accessor for the image data. The data returned is likely encoded in a specialized format.*
- virtual **Memory::uint8Array** [getRawData](#) () const =0  
*Accessor for the raw image data. The data returned should not be compressed or encoded.*
- virtual **Memory::uint8Array** [getRawGrayscaleData](#) (uint8\_t depth=8) const =0  
*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 std::shared\_ptr< [Image](#) > [openImage](#) (const uint8\_t \*data, const uint64\_t size)  
*Determine the image type of a buffer of image data and create an [Image](#) object.*
- static std::shared\_ptr< [Image](#) > [openImage](#) (const [Memory::uint8Array](#) &data)  
*Determine the image type of a buffer of image data and create an [Image](#) object.*
- static std::shared\_ptr< [Image](#) > [openImage](#) (const std::string &path)  
*Determine the image type of an image file and create an [Image](#) object.*
- static [CompressionAlgorithm](#) [getCompressionAlgorithm](#) (const uint8\_t \*data, const uint64\_t size)  
*Determine the compression algorithm of a buffer of image data.*
- static [CompressionAlgorithm](#) [getCompressionAlgorithm](#) (const [Memory::uint8Array](#) &data)  
*Determine the compression algorithm of a buffer of image data.*
- static [CompressionAlgorithm](#) [getCompressionAlgorithm](#) (const std::string &path)  
*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.*
- const uint8\_t \* [getDataPointer](#) () const
- uint64\_t [getDataSize](#) () const

### F.47.1 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.

### F.47.2 Constructor & Destructor Documentation

**BiometricEvaluation::Image::Image ( const uint8\_t \* data, const uint64\_t size, const [Size](#) dimensions, const uint32\_t depth, const [Resolution](#) resolution, const [CompressionAlgorithm](#) compression )**

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

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

**BiometricEvaluation::Image::Image ( const uint8\_t \* *data*, const uint64\_t *size*, const CompressionAlgorithm *compression* )**

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>compression</i>	The CompressionAlgorithm of data.

## Exceptions

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

### F.47.3 Member Function Documentation

**CompressionAlgorithm 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\(\)](#).

**static CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm ( const uint8\_t \* *data*, const uint64\_t *size* ) [static]**

Determine the compression algorithm of a buffer of image data.

## Parameters

in	<i>data</i>	The image data.
in	<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.

```
static CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm ( const  
Memory::uint8Array & data ) [static]
```

Determine the compression algorithm of a buffer of image data.

## Parameters

<i>in</i>	<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.

**static CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm ( const std::string & *path* ) [static]**

Determine the compression algorithm of a file.

## Parameters

<i>in</i>	<i>path</i>	Path to file.
-----------	-------------	---------------

## Returns

Compression algorithm used in the file.

## Exceptions

<a href="#">Error::ObjectDoesNotExist</a>	path does not exist.
<a href="#">Error::StrategyError</a>	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.

**Memory::uint8Array BiometricEvaluation::Image::Image::getData ( ) const**

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

## Returns

AutoArray holding image data.

**const uint8\_t\* BiometricEvaluation::Image::Image::getDataPointer ( ) const [protected]**

## Returns

Const pointer to buffer underlying `_data`.

**uint64\_t BiometricEvaluation::Image::Image::getDataSize ( ) const [protected]**

## Returns

[Size](#) of `_data`.

**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).

**Size BiometricEvaluation::Image::Image::getDimensions ( ) const**

Accessor for the dimensions of the image in pixels.

Returns

[Coordinate](#) object containing dimensions in pixels.

**virtual Memory::uint8Array BiometricEvaluation::Image::Image::getRawData ( ) const [pure virtual]**

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

Returns

AutoArray holding raw image data.

Exceptions

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

Implemented in [BiometricEvaluation::Image::NetPBM](#), [BiometricEvaluation::Image::JPEG2000](#), [BiometricEvaluation::Image::JPEG](#), [BiometricEvaluation::Image::PNG](#), [BiometricEvaluation::Image::Raw](#), [BiometricEvaluation::Image::BMP](#), [BiometricEvaluation::Image::JPEGL](#), and [BiometricEvaluation::Image::WSQ](#).

**virtual Memory::uint8Array BiometricEvaluation::Image::Image::getRawGrayscaleData ( uint8\_t depth = 8 ) const [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

AutoArray holding raw grayscale image data.

Exceptions

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

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implemented in [BiometricEvaluation::Image::NetPBM](#), [BiometricEvaluation::Image::JPEG2000](#), [BiometricEvaluation::Image::PNG](#), [BiometricEvaluation::Image::JPEG](#), [BiometricEvaluation::Image::Raw](#), [BiometricEvaluation::Image::BMP](#), [BiometricEvaluation::Image::WSQ](#), and [BiometricEvaluation::Image::JPEGL](#).

**Resolution** BiometricEvaluation::Image::Image::getResolution ( ) const

Accessor for the resolution of the image.

Returns

[Resolution](#) struct

**static** std::shared\_ptr<Image> BiometricEvaluation::Image::Image::openImage ( const uint8\_t \* data, const uint64\_t size ) [static]

Determine the image type of a buffer of image data and create an [Image](#) object.

Parameters

in	<i>data</i>	The image data.
in	<i>size</i>	The size of the image data, in bytes.

Returns

[Image](#) representation of the input data buffer.

Exceptions

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

**static** std::shared\_ptr<Image> BiometricEvaluation::Image::Image::openImage ( const Memory::uint8Array & data ) [static]

Determine the image type of a buffer of image data and create an [Image](#) object.

Parameters

in	<i>data</i>	The image data.
----	-------------	-----------------

Returns

[Image](#) representation of the input data buffer.

Exceptions

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

**static** std::shared\_ptr<Image> BiometricEvaluation::Image::Image::openImage ( const std::string & path ) [static]

Determine the image type of an image file and create an [Image](#) object.

Parameters

in	<i>path</i>	Path to image data.
----	-------------	---------------------

Returns

[Image](#) representation of the input data buffer.



Exceptions

<a href="#"><i>Error::DataError</i></a>	Error manipulating data.
<a href="#"><i>Error::ObjectDoesNotExist</i></a>	No file at specified path.
<a href="#"><i>Error::StrategyError</i></a>	Error while creating <a href="#">Image</a> .

**void BiometricEvaluation::Image::Image::setDepth ( const uint32\_t *depth* ) [protected]**

Mutator for the color depth of the image in bits.

Parameters

<i>in</i>	<i>depth</i>	The color depth of the image (bit).
-----------	--------------	-------------------------------------

**void BiometricEvaluation::Image::Image::setDimensions ( const Size *dimensions* ) [protected]**

Mutator for the dimensions of the image in pixels.

Parameters

<i>in</i>	<i>dimensions</i>	Dimensions of image (pixel).
-----------	-------------------	------------------------------

**void BiometricEvaluation::Image::Image::setResolution ( const Resolution *resolution* ) [protected]**

Mutator for the resolution of the image .

Parameters

<i>in</i>	<i>resolution</i>	<a href="#">Resolution</a> struct.
-----------	-------------------	------------------------------------

**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.

## F.47.4 Member Data Documentation

**const uint32\_t BiometricEvaluation::Image::Image::bitsPerComponent = 8 [static]**

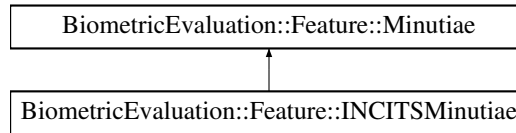
Number of bits per color component

## F.48 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](#) [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 std::string **FMR\_ANSI\_SPEC\_VERSION**
- static const std::string **FMR\_ISO\_SPEC\_VERSION**
- static const std::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

### F.48.1 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.

### F.48.2 Constructor & Destructor Documentation

**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.

### F.48.3 Member Function Documentation

**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.
----	------------	-------------------------

**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.
----	------------	-------------------------------

**void BiometricEvaluation::Feature::INCITSMinutiae::setMinutiaPoints ( const MinutiaPointSet & *mps* )**

Mutator for the minutiae point set.

Parameters

in	<i>mps</i>	The minutiae points.
----	------------	----------------------

**void BiometricEvaluation::Feature::INCITSMinutiae::setRidgeCountItems ( const RidgeCountItemSet & *rcis* )**

Mutator for the ridge count items.

Parameters

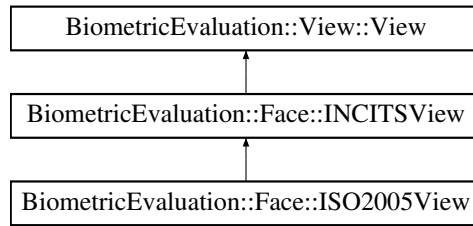
in	<i>rcis</i>	The set of ridge count items.
----	-------------	-------------------------------

## F.49 BiometricEvaluation::Face::INCITSView Class Reference

A class to represent single facial image view and derived information.

```
#include <be_face_incitsview.h>
```

Inheritance diagram for BiometricEvaluation::Face::INCITSView:



## Public Member Functions

- [Face::Gender](#) [getGender](#) () const  
*Obtain the gender.*
- [Face::EyeColor](#) [getEyeColor](#) () const  
*Obtain the eye color.*
- [Face::HairColor](#) [getHairColor](#) () const  
*Obtain the hair color.*
- bool [propertiesConsidered](#) () const  
*Indicate whether properties are specified.*
- void [getPropertySet](#) ([Face::PropertySet](#) &propertySet) const  
*Get the set of properties.*
- [BiometricEvaluation::Face::Expression](#) [getExpression](#) () const
- void [getFeaturePointSet](#) ([BiometricEvaluation::Feature::MPEGFacePointSet](#) &featurePointSet) const  
*Obtain the set of.*
- [Face::ImageType](#) [getImageType](#) () const  
*Obtain the face image type.*
- [Face::ImageDataType](#) [getImageDataType](#) () const  
*Obtain the face image data type.*
- [Face::PoseAngle](#) [getPoseAngle](#) () const  
*Obtain the face pose angle.*
- [Face::ColorSpace](#) [getColorSpace](#) () const  
*Obtain the color space.*
- [Face::SourceType](#) [getSourceType](#) () const  
*Obtain the source type.*
- uint16\_t [getDeviceType](#) () const  
*Obtain the device type.*

## Protected Member Functions

- [INCITSView](#) (const std::string &filename, const uint32\_t viewNumber)  
*Construct the common components of an INCITS face view from records contained in files.*
- [INCITSView](#) (const [Memory::uint8Array](#) &buffer, const uint32\_t viewNumber)  
*Construct an INCITS face view from a record contained in a buffer.*
- [Memory::uint8Array](#) const & [getFIDData](#) () const  
*Obtain a reference to the face image record data buffer.*
- virtual void [readHeader](#) ([BiometricEvaluation::Memory::IndexedBuffer](#) &buf, const uint32\_t format←  
Standard)

*Read the common face image data record header from an INCITS record, excepting the format identifier and version number data items.*

- virtual void [readFaceView](#) ([Memory::IndexedBuffer](#) &buf)

*Read the common face representation information from an INCITS record.*

## Static Protected Attributes

- static const uint32\_t **ISO2005\_STANDARD** = 1
- static const uint32\_t **BASE\_FORMAT\_ID** = 0x46414300

### F.49.1 Detailed Description

A class to represent single facial image view and derived information.

A base [Face::INCITSView](#) class represents an INCITS/ANSI or ISO face 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.

### F.49.2 Constructor & Destructor Documentation

**BiometricEvaluation::Face::INCITSView::INCITSView** ( const std::string &*filename*, const uint32\_t *viewNumber* ) **[protected]**

Construct the common components of an INCITS face view from records contained in files.

See documentation in child classes of INCITS for information on constructing INCITS-derived face views.

Parameters

in	<i>filename</i>	The name of the file containing the complete face image data record.
in	<i>viewNumber</i>	The eye number to use.

Exceptions

<a href="#">Error::DataError</a>	Invalid record format.
<a href="#">Error::FileError</a>	Could not open or read from file.

**BiometricEvaluation::Face::INCITSView::INCITSView** ( const Memory::uint8Array &*buffer*, const uint32\_t *viewNumber* ) **[protected]**

Construct an INCITS face view from a record contained in a buffer.

See documentation in child classes of INCITS for information on constructing INCITS-derived face views.

Parameters

in	<i>buffer</i>	The buffer containing the complete face image data record.
in	<i>viewNumber</i>	The eye number to use.

Exceptions

<a href="#">Error::DataError</a>	Invalid record format.
----------------------------------	------------------------

### F.49.3 Member Function Documentation

**Face::ColorSpace** **BiometricEvaluation::Face::INCITSView::getColorSpace** ( ) const

Obtain the color space.

Returns

The color space code.

**uint16\_t BiometricEvaluation::Face::INCITSView::getDeviceType ( ) const**

Obtain the device type.

Returns

The device type vendor code.

**Face::EyeColor BiometricEvaluation::Face::INCITSView::getEyeColor ( ) const**

Obtain the eye color.

Returns

The eye color code.

**void BiometricEvaluation::Face::INCITSView::getFeaturePointSet ( BiometricEvaluation::Feature←  
::MPEGFacePointSet & *featurePointSet* ) const**

Obtain the set of.  
Parameters

out	<i>featurePointSet</i>	The set of feature points.
-----	------------------------	----------------------------

**Memory::uint8Array const& BiometricEvaluation::Face::INCITSView::getFIDData ( ) const  
[protected]**

Obtain a reference to the face image record data buffer.

Returns

The entire face image record data.

**Face::Gender BiometricEvaluation::Face::INCITSView::getGender ( ) const**

Obtain the gender.

Returns

The gender code.

**Face::HairColor BiometricEvaluation::Face::INCITSView::getHairColor ( ) const**

Obtain the hair color.

Returns

The hair color code.

**Face::ImageDataType BiometricEvaluation::Face::INCITSView::getImageDataType ( ) const**

Obtain the face image data type.

Returns

The image data type.

**Face::ImageType BiometricEvaluation::Face::INCITSView::getImageType ( ) const**

Obtain the face image type.

Returns

The image type.

**Face::PoseAngle BiometricEvaluation::Face::INCITSView::getPoseAngle ( ) const**

Obtain the face pose angle.

Returns

The pose angle.

**void BiometricEvaluation::Face::INCITSView::getPropertySet ( Face::PropertySet & *propertySet* ) const**

Get the set of properties.

Returns

The set of properties.

**Face::SourceType BiometricEvaluation::Face::INCITSView::getSourceType ( ) const**

Obtain the source type.

Returns

The source type code.

**bool BiometricEvaluation::Face::INCITSView::propertiesConsidered ( ) const**

Indicate whether properties are specified.

Returns

true if properties are specified, false otherwise.

**virtual void BiometricEvaluation::Face::INCITSView::readFaceView ( Memory::IndexedBuffer & *buf* ) [protected], [virtual]**

Read the common face representation information from an INCITS record.

An [Face](#) representation from an INCITS record includes image information, gender, pose angle, etc.



## Parameters

<code>in, out</code>	<code>buf</code>	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the Facial information record.
----------------------	------------------	---

## Exceptions

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

```
virtual void BiometricEvaluation::Face::INCITSView::readHeader ( BiometricEvaluation↵  
::Memory::IndexedBuffer & buf, const uint32_t formatStandard ) [protected],  
[virtual]
```

Read the common face image data record header from an INCITS record, excepting the format identifier and version number data items.

## Parameters

<code>in</code>	<code>buf</code>	The indexed buffer containing the record data, with the index starting at the first octet after the format identifier and version number data items. The index of the buffer will be changed to the location after the header.
<code>in</code>	<code>formatStandard</code>	Value indicating which header version to read; must be ISO2005_STAN↵ DARD

## Exceptions

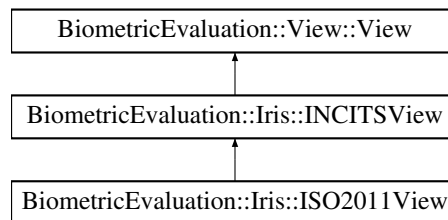
<i>ParameterError</i>	The formatStandard parameter is incorrect.
<i>DataError</i>	The INCITS record has invalid or missing data.

## F.50 BiometricEvaluation::Iris::INCITSView Class Reference

A class to represent single iris view and derived information.

```
#include <be_iris.incitsview.h>
```

Inheritance diagram for BiometricEvaluation::Iris::INCITSView:



### Classes

- struct [QualitySubBlock](#)

*Representation of an iris quality block.*

### Public Types

- typedef std::vector  
< [QualitySubBlock](#) > **QualitySet**

## Public Member Functions

- uint8\_t [getCertificationFlag](#) () const  
*Obtain the certification flag.*
- std::string [getCaptureDateString](#) () const  
*Obtain the capture date as a string.*
- [Iris::CaptureDeviceTechnology](#) [getCaptureDeviceTechnology](#) () const  
*Obtain the capture device technology.*
- uint16\_t [getCaptureDeviceVendor](#) () const  
*Obtain the capture device vendor.*
- uint16\_t [getCaptureDeviceType](#) () const  
*Obtain the capture device type.*
- void [getQualitySet](#) (Iris::INCITSView::QualitySet &qualitySet) const  
*Obtain the set of quality sub-blocks.*
- [Iris::EyeLabel](#) [getEyeLabel](#) () const  
*Obtain the eye label type.*
- [Iris::ImageType](#) [getImageType](#) () const  
*Obtain the iris image type.*
- void [getImageProperties](#) (BiometricEvaluation::Iris::Orientation &horizontalOrientation, BiometricEvaluation←  
::Iris::Orientation &verticalOrientation, BiometricEvaluation←  
::ImageCompression &compression←  
History) const  
*Obtain the iris image properties.*
- uint16\_t [getCameraRange](#) ()  
*Obtain the camera range.*
- void [getRollAngleInfo](#) (uint16\_t &rollAngle, uint16\_t &rollAngleUncertainty)  
*Obtain the roll angle information.*
- void [getIrisCenterInfo](#) (uint16\_t &irisCenterSmallestX, uint16\_t &irisCenterSmallestY, uint16\_t &iris←  
CenterLargestX, uint16\_t &irisCenterLargestY, uint16\_t &irisDiameterSmallest, uint16\_t &irisDiameter←  
Largest)  
*Obtain the iris center information. COORDINATE\_UNDEF may be returned for any of the out parameters.*

## Static Public Attributes

- static const uint16\_t **RANGE\_UNASSIGNED** = 0
- static const uint16\_t **RANGE\_FAILED** = 1
- static const uint16\_t **RANGE\_OVERFLOW** = 65535
- static const uint16\_t **ROLL\_ANGLE\_UNDEF** = 65535
- static const uint16\_t **ROLL\_UNCERTAIN\_UNDEF** = 65535
- static const uint16\_t **COORDINATE\_UNDEF** = 0

## Protected Member Functions

- [INCITSView](#) (const std::string &filename, const uint32\_t viewNumber)  
*Construct the common components of an INCITS iris view from records contained in files.*
- [INCITSView](#) (const [Memory::uint8Array](#) &buffer, const uint32\_t viewNumber)  
*Construct an INCITS iris view from a record contained in a buffer.*
- [Memory::uint8Array](#) const & [getIIRData](#) () const



Exceptions

<a href="#"><i>Error::DataError</i></a>	Invalid record format.
---	------------------------

### F.50.3 Member Function Documentation

**uint16\_t BiometricEvaluation::Iris::INCITSView::getCameraRange ( )**

Obtain the camera range.

RANGE\_UNASSIGNED, RANGE\_FAILED, or RANGE\_OVERFLOW may be returned.

Returns

The camera range.

**std::string BiometricEvaluation::Iris::INCITSView::getCaptureDateString ( ) const**

Obtain the capture date as a string.

Returns

The capture data and time.

**Iris::CaptureDeviceTechnology BiometricEvaluation::Iris::INCITSView::getCaptureDeviceTechnology ( ) const**

Obtain the capture device technology.

Returns

The capture device technology identifier.

**uint16\_t BiometricEvaluation::Iris::INCITSView::getCaptureDeviceType ( ) const**

Obtain the capture device type.

Returns

The capture device type ID.

**uint16\_t BiometricEvaluation::Iris::INCITSView::getCaptureDeviceVendor ( ) const**

Obtain the capture device vendor.

Returns

The capture device vendor ID.

**uint8\_t BiometricEvaluation::Iris::INCITSView::getCertificationFlag ( ) const**

Obtain the certification flag.

Returns

The certification flag.

**Iris::EyeLabel BiometricEvaluation::Iris::INCITSView::getEyeLabel ( ) const**

Obtain the eye label type.

Returns

The eye label.

**Memory::uint8Array const& BiometricEvaluation::Iris::INCITSView::getIIRData ( ) const  
[protected]**

Obtain a reference to the iris image record data buffer.

Returns

The entire iris image record data.

**void BiometricEvaluation::Iris::INCITSView::getImageProperties ( BiometricEvaluation::Iris::↔  
Orientation & *horizontalOrientation*, BiometricEvaluation::Iris::Orientation & *verticalOrientation*,  
BiometricEvaluation::Iris::ImageCompression & *compressionHistory* ) const**

Obtain the iris image properties.

Parameters

out	<i>horizontal↔ Orientation</i>	The horizontal orientation.
out	<i>vertical↔ Orientation</i>	The vertical orientation.
out	<i>compression↔ History</i>	The image compression history.

**Iris::ImageType BiometricEvaluation::Iris::INCITSView::getImageType ( ) const**

Obtain the iris image type.

Returns

The image type.

**void BiometricEvaluation::Iris::INCITSView::getIrisCenterInfo ( uint16\_t & *irisCenterSmallestX*,  
uint16\_t & *irisCenterSmallestY*, uint16\_t & *irisCenterLargestX*, uint16\_t & *irisCenterLargestY*, uint16\_t  
& *irisDiameterSmallest*, uint16\_t & *irisDiameterLargest* )**

Obtain the iris center information. COORDINATE\_UNDEF may be returned for any of the out parameters.

Parameters

out	<i>irisCenter↔ SmallestX</i>	Smallest expected iris center X coordinate in pixels.
-----	----------------------------------	---

out	<i>irisCenter</i> ↔ <i>SmallestY</i>	Smallest expected iris center Y coordinate in pixels.
out	<i>irisCenter</i> ↔ <i>LargestX</i>	Largest expected iris center X coordinate in pixels.
out	<i>irisCenter</i> ↔ <i>LargestY</i>	Largest expected iris center Y coordinate in pixels.
out	<i>irisDiameter</i> ↔ <i>Smallest</i>	Smallest expected iris diameter in pixels.
out	<i>irisDiameter</i> ↔ <i>Largest</i>	Largest expected iris diameter in pixels.

**void BiometricEvaluation::Iris::INCITSView::getQualitySet ( Iris::INCITSView::QualitySet & *qualitySet* ) const**

Obtain the set of quality sub-blocks.

Parameters

out	<i>qualitySet</i>	The set of quality sub-blocks.
-----	-------------------	--------------------------------

**void BiometricEvaluation::Iris::INCITSView::getRollAngleInfo ( uint16\_t & *rollAngle*, uint16\_t & *rollAngleUncertainty* )**

Obtain the roll angle information.

Parameters

out	<i>rollAngle</i>	The roll angle.
out	<i>rollAngle</i> ↔ <i>Uncertainty</i>	The roll angle uncertainty.

**virtual void BiometricEvaluation::Iris::INCITSView::readHeader ( BiometricEvaluation↔  
::Memory::IndexedBuffer & *buf*, const uint32\_t *formatStandard* ) [protected],  
[virtual]**

Read the common iris image record header from an INCITS record, excepting the format identifier and version number data items.

Parameters

in	<i>buf</i>	The indexed buffer containing the record data, with the index starting at the first octet after the format identifier and version number data items. 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; must be ISO2011_STAN↔ DARD

Exceptions

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

**virtual void BiometricEvaluation::Iris::INCITSView::readIrisView ( Memory::IndexedBuffer & *buf* ) [protected], [virtual]**

Read the common iris representation information from an INCITS record.

An [Iris](#) Representation from an INCITS record includes image information, cropping information, etc.

## Parameters

<code>in, out</code>	<code>buf</code>	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the <a href="#">Iris</a> Representation.
----------------------	------------------	---

## Exceptions

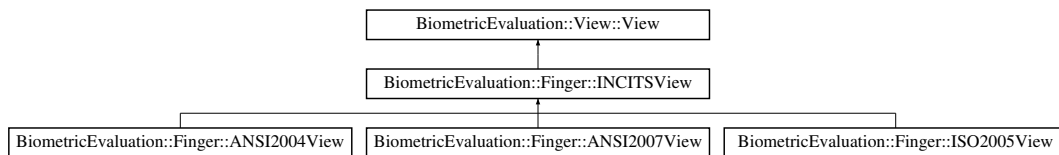
<code>DataError</code>	The INCITS record has invalid or missing data.
------------------------	--

## F.51 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](#) `getPosition ()` const  
*Obtain the finger position.*
- [Finger::Impression](#) `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'.*
- `std::shared_ptr< Image::Image >` `getImage ()` const

### Static Public Member Functions

- static [Finger::Position](#) `convertPosition (int incitsFGP)`  
*Convert a finger position code from an INCITS finger record to the common code.*
- static [Finger::Impression](#) `convertImpression (int incitsIMP)`  
*Convert a impression type code from an INCITS finger record to the common code.*



## Protected Member Functions

- [INCITSView](#) (const std::string &fmrFilename, const std::string &firFilename, const uint32\_t view↵  
Number)  
*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)  
*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](#) &position)  
*Mutator for the position.*
- void [setImpressionType](#) (const [Finger::Impression](#) &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 [readFMRHeader](#) ([Memory::IndexedBuffer](#) &buf, const uint32\_t formatStandard)  
*Read the common finger minutiae record header from an INCITS record.*
- void [readFVMR](#) ([Memory::IndexedBuffer](#) &buf)  
*Read the common finger view record information from an INCITS record.*
- virtual [Feature::MinutiaPointSet](#) [readMinutiaeDataPoints](#) ([Memory::IndexedBuffer](#) &buf, uint32\_t count)  
*Read the minutiae data points, and extended data blocks.*
- virtual void [readExtendedDataBlock](#) ([Memory::IndexedBuffer](#) &buf)  
*Read the common extended data block.*
- virtual [Feature::RidgeCountItemSet](#) [readRidgeCountData](#) ([Memory::IndexedBuffer](#) &buf, uint32\_t data↵  
Length)  
*Read the ridge count data.*
- virtual void [readCoreDeltaData](#) ([Memory::IndexedBuffer](#) &buf, uint32\_t dataLength, [Feature::Core↵  
PointSet](#) &cores, [Feature::DeltaPointSet](#) &deltas)=0  
*Read the core points data.*

## Static Protected Attributes

- static const uint32\_t **FMR\_BASE\_FORMAT\_ID** = 0x464D5200
- static const uint32\_t **ANSI2004\_STANDARD** = 1

*The type of record that will be read by the subclass.*

- static const uint32\_t **ISO2005\_STANDARD** = 2
- static const uint32\_t **ANSI2007\_STANDARD** = 3

### F.51.1 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.

### F.51.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::INCITSView::INCITSView ( const std::string &fmrFilename, const std::string &firFilename, const uint32\_t viewNumber ) [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

<a href="#">Error::DataError</a>	Invalid record format.
<a href="#">Error::FileError</a>	Could not open or read from file.

**BiometricEvaluation::Finger::INCITSView::INCITSView ( const Memory::uint8Array &fmrBuffer, const Memory::uint8Array &firBuffer, const uint32\_t viewNumber ) [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

<a href="#">Error::DataError</a>	Invalid record format.
----------------------------------	------------------------

### F.51.3 Member Function Documentation

**static Finger::Impression BiometricEvaluation::Finger::INCITSView::convertImpression ( int *incitsIMP* ) [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

<a href="#"><i>Error::DataError</i></a>	The impression type code is invalid.
---	--------------------------------------

## Returns

The finger impression type code in common notation.

**static Finger::Position BiometricEvaluation::Finger::INCITSView::convertPosition ( int *incitsFGP* )  
[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

<a href="#"><i>Error::DataError</i></a>	The position code is invalid.
---	-------------------------------

## Returns

The finger position code in common notation.

**uint16\_t BiometricEvaluation::Finger::INCITSView::getCaptureEquipmentID ( ) const**

Obtain the capture equipment identifier.

## Returns

The equipment ID.

**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.

**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.

**Finger::Impression BiometricEvaluation::Finger::INCITSView::getImpressionType ( ) const**

Obtain the finger impression code.

Returns

The finger impression code.

**Finger::Position BiometricEvaluation::Finger::INCITSView::getPosition ( ) const**

Obtain the finger position.

Returns

The finger position.

**uint32\_t BiometricEvaluation::Finger::INCITSView::getQuality ( ) const**

Obtain the finger quality value.

Returns

The finger quality value.

**bool BiometricEvaluation::Finger::INCITSView::isAppendixFCompliant ( ) const**

Obtain the capture equipment compliance indicator for 'Appendix F'.

Returns

True if 'Appendix F' compliant, false otherwise.

**virtual void BiometricEvaluation::Finger::INCITSView::readCoreDeltaData ( Memory::IndexedBuffer & *buf*, uint32\_t *dataLength*, Feature::CorePointSet & *cores*, Feature::DeltaPointSet & *deltas* ) [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](#).

**virtual void BiometricEvaluation::Finger::INCITSView::readExtendedDataBlock ( Memory::IndexedBuffer & *buf* ) [protected], [virtual]**

Read the common extended data block.

## Parameters

<i>in, out</i>	<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.
------------------	--

**void BiometricEvaluation::Finger::INCITSView::readFMRHeader ( Memory::IndexedBuffer & buf, const uint32\_t formatStandard ) [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

<i>in</i>	<i>buf</i>	The indexed buffer containing the record data. The index must start after the Format ID and spec version fields in the header. The index of the buffer will be changed to the location after the header.
<i>in</i>	<i>formatStandard</i>	Value indicating which header version to read; one of ANSI2004_STANDARD or ISO2005_STANDARD.

## Exceptions

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

**void BiometricEvaluation::Finger::INCITSView::readFVMR ( Memory::IndexedBuffer & buf ) [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

<i>in, out</i>	<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.
------------------	--

**virtual Feature::MinutiaPointSet BiometricEvaluation::Finger::INCITSView::readMinutiaeDataPoints ( Memory::IndexedBuffer & buf, uint32\_t count ) [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.
------------------	--

**virtual Feature::RidgeCountItemSet BiometricEvaluation::Finger::INCITSView::read↵  
RidgeCountData ( Memory::IndexedBuffer & *buf*, uint32\_t *dataLength* ) [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.

**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.
----	-------------	--

**void BiometricEvaluation::Finger::INCITSView::setCaptureEquipmentID ( uint16\_t *id* )  
[protected]**

Mutator for the equipment ID.

## Parameters

in	<i>id</i>	The equipment ID value.
----	-----------	-------------------------

**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.

**void BiometricEvaluation::Finger::INCITSView::setImpressionType ( const Finger::Impression & *impression* ) [protected]**

Mutator for the impression type.

Parameters

<code>in</code>	<code>impression</code>	The finger impression type code.
-----------------	-------------------------	----------------------------------

**void BiometricEvaluation::Finger::INCITSView::setMinutiaeData ( const Feature::INCITSMinutiae & *fmd* ) [protected]**

Mutator for the [Feature::INCITSMinutiae](#) item.

Parameters

<code>in</code>	<code>fmd</code>	The minutiae data object.
-----------------	------------------	---------------------------

**void BiometricEvaluation::Finger::INCITSView::setPosition ( const Finger::Position & *position* ) [protected]**

Mutator for the position.

Parameters

<code>in</code>	<code>position</code>	The finger position.
-----------------	-----------------------	----------------------

**void BiometricEvaluation::Finger::INCITSView::setQuality ( uint32\_t *quality* ) [protected]**

Mutator for the finger quality value.

Parameters

<code>in</code>	<code>quality</code>	The quality value.
-----------------	----------------------	--------------------

**void BiometricEvaluation::Finger::INCITSView::setViewNumber ( uint32\_t *viewNumber* ) [protected]**

Mutator for the finger view number.

Parameters

<code>in</code>	<code>viewNumber</code>	The view number value.
-----------------	-------------------------	------------------------

## F.52 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)

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)

Set the current index into the buffer.

- [uint8\\_t](#) [scanU8Val](#) ()

Obtain the next element of the buffer and increment the current index value.

- [uint16\\_t](#) [scanU16Val](#) ()

Obtain the next two elements of the buffer and increment the current index value.

- [uint16\\_t](#) [scanBeU16Val](#) ()

Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.

- [uint32\\_t](#) [scanU32Val](#) ()

Obtain the next four elements of the buffer and increment the current index value by four.

- [uint32\\_t](#) [scanBeU32Val](#) ()

Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.

- [uint64\\_t](#) [scanU64Val](#) ()

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)

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.

### F.52.1 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 and 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.

### F.52.2 Constructor & Destructor Documentation

**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.

### F.52.3 Member Function Documentation

#### uint32\_t BiometricEvaluation::Memory::IndexedBuffer::getIndex ( )

Obtain the current index into the buffer.

Returns

The current buffer index.

#### uint32\_t BiometricEvaluation::Memory::IndexedBuffer::getSize ( )

Obtain the current size of the buffer.

Returns

The current buffer size.

#### 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

<i>i</i>	<i>i</i>	The subscript.
----------	----------	----------------

Returns

Reference to element '*i*' of the buffer.

#### 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

<i>i</i>	<i>i</i>	The subscript.
----------	----------	----------------

Returns

Reference to const element '*i*' of the buffer.

#### uint32\_t BiometricEvaluation::Memory::IndexedBuffer::scan ( void \* buf, const uint32\_t len )

Obtain the next '*n*' elements of the buffer and increment the current index value by *n*.

Parameters

<i>i</i>	<i>buf</i>	Buffer to store the copied data. Can be nullptr. The current index is incremented.
----------	------------	--

<i>in</i>	<i>len</i>	The number of elements to copy.
-----------	------------	---------------------------------

Exceptions

<a href="#"><i>Error::DataError</i></a>	The buffer is exhausted.
---	--------------------------

Returns

The number of elements copied.

#### **uint16\_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU16Val ( )**

Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.

Exceptions

<a href="#"><i>Error::DataError</i></a>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 16-bit value.

#### **uint32\_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU32Val ( )**

Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.

Exceptions

<a href="#"><i>Error::DataError</i></a>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 32-bit value.

#### **uint16\_t BiometricEvaluation::Memory::IndexedBuffer::scanU16Val ( )**

Obtain the next two elements of the buffer and increment the current index value.

Exceptions

<a href="#"><i>Error::DataError</i></a>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 16-bit value.

#### **uint32\_t BiometricEvaluation::Memory::IndexedBuffer::scanU32Val ( )**

Obtain the next four elements of the buffer and increment the current index value by four.

Exceptions

<a href="#"><i>Error::DataError</i></a>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 32-bit value.

**uint64\_t BiometricEvaluation::Memory::IndexedBuffer::scanU64Val ( )**

Obtain the next eight elements of the buffer and increment the current index value by eight.

Exceptions

<a href="#"><i>Error::DataError</i></a>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 64-bit value.

**uint8\_t BiometricEvaluation::Memory::IndexedBuffer::scanU8Val ( )**

Obtain the next element of the buffer and increment the current index value.

Exceptions

<a href="#"><i>Error::DataError</i></a>	The buffer is exhausted.
---	--------------------------

Returns

The next element of the buffer as an unsigned 8-bit value.

**void BiometricEvaluation::Memory::IndexedBuffer::setIndex ( uint32\_t index )**

Set the current index into the buffer.

Parameters

in	<i>index</i>	The index value to set.
----	--------------	-------------------------

Exceptions

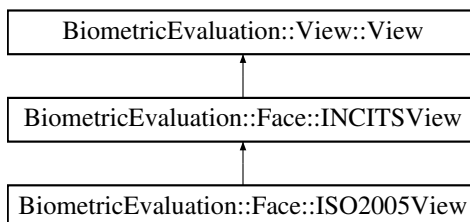
<a href="#"><i>Error::ParameterError</i></a>	The index parameter is too large.
--	-----------------------------------

## F.53 BiometricEvaluation::Face::ISO2005View Class Reference

A class to represent single face view and derived information.

```
#include <be_face_iso2005view.h>
```

Inheritance diagram for BiometricEvaluation::Face::ISO2005View:



## Public Member Functions

- [ISO2005View](#) ()  
*Construct an empty ISO2005 [Face Image](#) Data record.*
- [ISO2005View](#) (const std::string &filename, const uint32\_t viewNumber)  
*Construct an ISO 2005 face view from the named file.*
- [ISO2005View](#) (const [Memory::uint8Array](#) &buffer, const uint32\_t viewNumber)  
*Construct an ISO 2005 face view from a record contained in a buffer.*

## Protected Member Functions

- void [readISOHeader](#) ([BiometricEvaluation::Memory::IndexedBuffer](#) &buf)  
*Read the face image data record header from an ISO 2005 record.*

## Static Protected Attributes

- static const uint32\_t **BASE\_SPEC\_VERSION** = 0x30313000

### F.53.1 Detailed Description

A class to represent single face view and derived information.

A base [Face::ISO2005View](#) class represents an ISO 2005 face image data view.

### F.53.2 Constructor & Destructor Documentation

**BiometricEvaluation::Face::ISO2005View::ISO2005View** ( const std::string & *filename*, const uint32\_t *viewNumber* )

Construct an ISO 2005 face view from the named file.

The entire face image data record is passed into this method, with the specific instance of the facial image that is to be extracted from the record.

Parameters

in	<i>filename</i>	The name of the file containing the complete face image data record.
in	<i>viewNumber</i>	The facial information instance to read.

Exceptions

<a href="#">Error::DataError</a>	Invalid record format.
<a href="#">Error::FileError</a>	Could not open or read from file.

**BiometricEvaluation::Face::ISO2005View::ISO2005View** ( const [Memory::uint8Array](#) & *buffer*, const uint32\_t *viewNumber* )

Construct an ISO 2005 face view from a record contained in a buffer.

The entire face image data record is passed into this method, with the specific instance of the facial image that is to be extracted from the record.

Parameters

in	<i>buffer</i>	The buffer containing the complete face image data record.
in	<i>viewNumber</i>	The facial information instance to read.

Exceptions

<a href="#"><i>Error::DataError</i></a>	Invalid record format.
---	------------------------

### F.53.3 Member Function Documentation

**void BiometricEvaluation::Face::ISO2005View::readISOHeader ( BiometricEvaluation::Memory::↵ IndexedBuffer & *buf* ) [protected]**

Read the face image data record header from an ISO 2005 record.

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.
----	------------	--

Exceptions

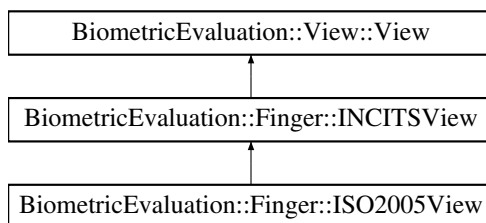
<i>DataError</i>	The record has invalid or missing data.
------------------	---

## F.54 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)

*Construct an ISO-2005 finger view from records contained in files.*

- [ISO2005View](#) (Memory::uint8Array &fmrBuffer, [Memory::uint8Array](#) &firBuffer, const uint32\_t view↵ Number)

*Construct an ISO-2005 finger view from records contained in buffers.*

### Protected Member Functions

- void **readFMRHeader** ([Memory::IndexedBuffer](#) &buf)
- void **readCoreDeltaData** ([Memory::IndexedBuffer](#) &buf, uint32\_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)

*Read the core points data.*

## Static Protected Attributes

- static const uint32\_t **BASE\_SPEC\_VERSION** = 0x20323000

## Additional Inherited Members

### F.54.1 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.

### F.54.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::ISO2005View::ISO2005View ( const std::string & *fmrFilename*, const std::string & *firFilename*, const uint32\_t *viewNumber* )**

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.

**BiometricEvaluation::Finger::ISO2005View::ISO2005View ( Memory::uint8Array & *fmrBuffer*, Memory::uint8Array & *firBuffer*, const uint32\_t *viewNumber* )**

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

<a href="#">Error::DataError</a>	Invalid record format.
----------------------------------	------------------------

### F.54.3 Member Function Documentation

**void BiometricEvaluation::Finger::ISO2005View::readCoreDeltaData ( Memory::IndexedBuffer & *buf*, uint32\_t *dataLength*, Feature::CorePointSet & *cores*, Feature::DeltaPointSet & *deltas* )**  
**[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.

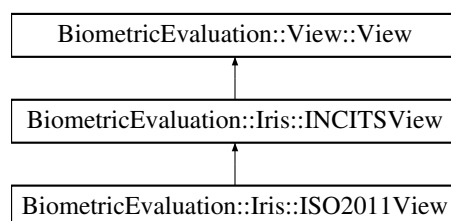
Implements [BiometricEvaluation::Finger::INCITSView](#).

## F.55 BiometricEvaluation::Iris::ISO2011View Class Reference

A class to represent single iris view and derived information.

```
#include <be_iris_iso2011view.h>
```

Inheritance diagram for BiometricEvaluation::Iris::ISO2011View:



### Public Member Functions

- [ISO2011View](#) ()  
*Construct an empty ISO 2011 iris view.*
- [ISO2011View](#) (const std::string &filename, const uint32\_t viewNumber)  
*Construct an ISO 2011 iris view from the named file.*
- [ISO2011View](#) (const [Memory::uint8Array](#) &buffer, const uint32\_t viewNumber)  
*Construct an ISO 2011 iris view from a record contained in a buffer.*

### Protected Member Functions

- void **readISOHeader** ([BiometricEvaluation::Memory::IndexedBuffer](#) &buf)

### Static Protected Attributes

- static const uint32\_t **BASE\_SPEC\_VERSION** = 0x30323000

### Additional Inherited Members

#### F.55.1 Detailed Description

A class to represent single iris view and derived information.

An Iris::ISO2011VIEW class represents an ISO 19794-6 iris image record view.



### F.55.2 Constructor & Destructor Documentation

**BiometricEvaluation::Iris::ISO2011View::ISO2011View** ( `const std::string &filename`, `const uint32_t viewNumber` )

Construct an ISO 2011 iris view from the named file.

## Parameters

in	<i>filename</i>	The name of the file containing the complete iris image record.
in	<i>viewNumber</i>	The eye number to use.

## Exceptions

<a href="#"><i>Error::DataError</i></a>	Invalid record format.
<a href="#"><i>Error::FileError</i></a>	Could not open or read from file.

**BiometricEvaluation::Iris::ISO2011View::ISO2011View** ( const Memory::uint8Array & *buffer*, const uint32\_t *viewNumber* )

Construct an ISO 2011 iris view from a record contained in a buffer.

## Parameters

in	<i>buffer</i>	The buffer containing the complete iris image record.
in	<i>viewNumber</i>	The eye number to use.

## Exceptions

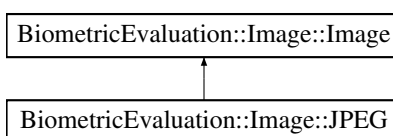
<a href="#"><i>Error::DataError</i></a>	Invalid record format.
---	------------------------

## F.56 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)
- [Memory::uint8Array getRawGrayscaleData](#) (uint8\_t depth=8) const  
*Accessor for decompressed data in grayscale.*
- [Memory::uint8Array getRawData](#) () const  
*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, uint64\_t size)
- static int [getc\\_skip\\_marker\\_segment](#) (const unsigned short marker, unsigned char \*\*cbufptr, unsigned char \*ebufptr)

## Additional Inherited Members

### F.56.1 Detailed Description

A JPEG-encoded image.

### F.56.2 Member Function Documentation

**Memory::uint8Array BiometricEvaluation::Image::JPEG::getRawData ( ) const [virtual]**

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

Returns

AutoArray holding raw image data.

Exceptions

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

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

**Memory::uint8Array BiometricEvaluation::Image::JPEG::getRawGrayscaleData ( uint8\_t *depth* = 8 ) const [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

AutoArray holding raw grayscale image data.

Exceptions

<a href="#"><i>Error::DataError</i></a>	<a href="#">Error</a> decompressing image data.
<a href="#"><i>Error::ParameterError</i></a>	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. *depth* adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

**static bool BiometricEvaluation::Image::JPEG::isJPEG ( const uint8\_t \* *data*, uint64\_t *size* ) [static]**

Whether or not data is a Lossy [JPEG](#) image.

Parameters

in	data	The buffer to check.
in	size	The size of data.

Returns

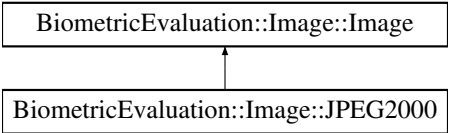
true if data appears to be a Lossy JPEG image, false otherwise

F.57 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)  
Create a new JPEG2000 object.
- Memory::uint8Array getRawData () const  
Accessor for the raw image data. The data returned should not be compressed or encoded.
- Memory::uint8Array getRawGrayscaleData (uint8\_t depth=8) const  
Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isJPEG2000 (const uint8\_t \*data, uint64\_t size)

Additional Inherited Members

F.57.1 Detailed Description

A JPEG-2000-encoded image.

F.57.2 Constructor & Destructor Documentation

BiometricEvaluation::Image::JPEG2000::JPEG2000 ( const uint8\_t \* data, const uint64\_t size, const int8\_t codec = 2 )

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 OPJ_CODEC_FORMAT used to encode data.

## Exceptions

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

**F.57.3 Member Function Documentation**

**Memory::uint8Array BiometricEvaluation::Image::JPEG2000::getRawData ( ) const [virtual]**

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

## Returns

AutoArray holding raw image data.

## Exceptions

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

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

**Memory::uint8Array BiometricEvaluation::Image::JPEG2000::getRawGrayscaleData ( uint8\_t depth = 8 ) const [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

AutoArray holding raw grayscale image data.

## Exceptions

<a href="#"><i>Error::DataError</i></a>	Error decompressing image data.
<a href="#"><i>Error::ParameterError</i></a>	Invalid value for depth.

## Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

**static bool BiometricEvaluation::Image::JPEG2000::isJPEG2000 ( const uint8\_t \* data, uint64\_t size ) [static]**

Whether or not data is a JPEG-2000 image.

Parameters

in	data	The buffer to check.
in	size	The size of data.

Returns

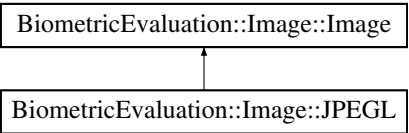
true if data appears to be a JPEG-2000 image, false otherwise.

F.58 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)
- [Memory::uint8Array getRawGrayscaleData](#) (uint8\_t depth=8) const  
*Accessor for decompressed data in grayscale.*
- [Memory::uint8Array getRawData](#) () const  
*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, uint64\_t size)

Additional Inherited Members

F.58.1 Detailed Description

A Lossless JPEG-encoded image.

F.58.2 Member Function Documentation

**Memory::uint8Array BiometricEvaluation::Image::JPEGL::getRawData ( ) const [virtual]**

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

Returns

AutoArray holding raw image data.

Exceptions

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

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

**Memory::uint8Array BiometricEvaluation::Image::JPEG::getRawGrayscaleData ( uint8\_t *depth* = 8 ) const [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

AutoArray holding raw grayscale image data.

Exceptions

<a href="#"><i>Error::DataError</i></a>	Error decompressing image data.
<a href="#"><i>Error::ParameterError</i></a>	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. *depth* adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

**static bool BiometricEvaluation::Image::JPEG::isJPEG ( const uint8\_t \* *data*, uint64\_t *size* ) [static]**

Whether or not data is a Lossless [JPEG](#) image.

Parameters

<i>in</i>	<i>data</i>	The buffer to check.
<i>in</i>	<i>size</i>	The size of data.

Returns

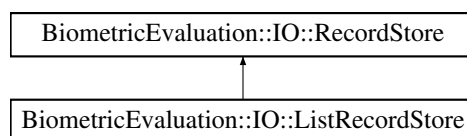
true if data appears to be a Lossless [JPEG](#) image, false otherwise.

## F.59 BiometricEvaluation::IO::ListRecordStore Class Reference

[RecordStore](#) that reads a list of keys from a text file, and retrieves the data from another [RecordStore](#).

```
#include <be_io_listrecstore.h>
```

Inheritance diagram for BiometricEvaluation::IO::ListRecordStore:



## Public Member Functions

- [ListRecordStore](#) (const std::string &pathname)
- [~ListRecordStore](#) ()
- void [insert](#) (const std::string &key, const void \*const data, const uint64\_t size)
- void [remove](#) (const std::string &key)
- uint64\_t [read](#) (const std::string &key, void \*const data) const
- void [replace](#) (const std::string &key, const void \*const data, const uint64\_t size)
- uint64\_t [length](#) (const std::string &key) const
- void [flush](#) (const std::string &key) const
- void [sync](#) () const
- uint64\_t [sequence](#) (std::string &key, void \*const data=nullptr, int cursor=BE\_RECSTORE\_SEQ\_NEXT)  
*Sequence through a [RecordStore](#), returning the key/data pairs.*
- void [setCursorAtKey](#) (const std::string &key)
- void [move](#) (const std::string &pathname)  
*Move the [RecordStore](#).*
- uint64\_t [getSpaceUsed](#) () const  
*Obtain real storage utilization.*

## Static Public Attributes

- static const std::string [SOURCERECORDSTOREPROPERTY](#)
- static const std::string [KEYLISTFILENAME](#)

## Additional Inherited Members

### F.59.1 Detailed Description

[RecordStore](#) that reads a list of keys from a text file, and retrieves the data from another [RecordStore](#).

ListRecordStores must be hand-crafted by first setting the 'Source Record Store', 'Type', and 'Count' properties in the .rscontrol.prop file. 'Source Record Store' is the complete path of the [RecordStore](#) containing the actual data records. Type must be 'List'. Count should match the number of entries in the file created next. Other properties are as in a "normal" [RecordStore](#); see example below.

Second, create a file called 'KeyList.txt' in the [RecordStore](#) directory containing a list of keys, one per line.

ListRecordStores can also be created and modified with versions of rstool(1) from 2013 or later.

Example .rscontrol.prop file: Count = 10 Description = Search records for SDK TESTSDK Name = Test↵  
 LRS Type = List Source Record Store = /Users/wsalamon/sandbox/SD29.rs

Note

List RecordStores must be opened read-only.

### F.59.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::ListRecordStore::ListRecordStore ( const std::string & pathname )**

Constructor, always opening read-only

**BiometricEvaluation::IO::ListRecordStore::~~ListRecordStore ( )**

Destructor



### F.59.3 Member Function Documentation

**void BiometricEvaluation::IO::ListRecordStore::flush ( const std::string & *key* ) const** **[virtual]**

Commit the record's data to storage.

## Parameters

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

## Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::ListRecordStore::getSpaceUsed ( ) const [virtual]**

Obtain real storage utilization.

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

## Returns

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

## Exceptions

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

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

**void BiometricEvaluation::IO::ListRecordStore::insert ( const std::string & key, const void \*const data, const uint64\_t size ) [virtual]**

Insert a record into the store.

## Parameters

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

## Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::ListRecordStore::length ( const std::string & key ) const [virtual]**

Return the length of a record.

## Parameters

in	key	The key of the record.
----	-----	------------------------

## Returns

The record length.

Exceptions

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

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

**void BiometricEvaluation::IO::ListRecordStore::move ( const std::string & *pathname* ) [virtual]**

Move the [RecordStore](#).

The [RecordStore](#) can be moved to a new path in the file system.

Parameters

in	<i>pathname</i>	The new path of the <a href="#">RecordStore</a> .
----	-----------------	---

Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::ListRecordStore::read ( const std::string & *key*, void \*const *data* ) const [virtual]**

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

Parameters

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

Returns

The size of the record.

Exceptions

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

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

**void BiometricEvaluation::IO::ListRecordStore::remove ( const std::string & *key* ) [virtual]**

Remove a record from the store.

Parameters

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

Exceptions

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

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

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

**void BiometricEvaluation::IO::ListRecordStore::replace ( const std::string & key, const void \*const data, const uint64\_t size ) [virtual]**

Replace a complete record in a store.

Parameters

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

Exceptions

<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](#).

**uint64\_t BiometricEvaluation::IO::ListRecordStore::sequence ( std::string & key, void \*const data = nullptr, int cursor = BE\_RECSTORE\_SEQ\_NEXT ) [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 nullptr 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](#).

**void BiometricEvaluation::IO::ListRecordStore::setCursorAtKey ( const std::string & key ) [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

<code>in</code>	<code>key</code>	The key of the record which will be returned by the first subsequent call to <a href="#">sequence()</a> .
-----------------	------------------	---

## Exceptions

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

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

**void BiometricEvaluation::IO::ListRecordStore::sync ( ) const [virtual]**

Synchronize the entire record store to persistent storage.

## Exceptions

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

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

## F.59.4 Member Data Documentation

**const std::string BiometricEvaluation::IO::ListRecordStore::KEYLISTFILENAME [static]**

File name containing the list of keys

**const std::string BiometricEvaluation::IO::ListRecordStore::SOURCERECORDSTOREPROPERTY [static]**

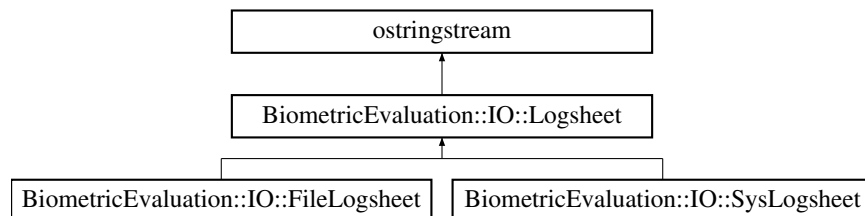
Property key for the source [RecordStore](#)

## F.60 BiometricEvaluation::IO::Logsheet Class Reference

A class to represent a logging mechanism.

```
#include <be_io_logsheet.h>
```

Inheritance diagram for BiometricEvaluation::IO::Logsheet:



### Public Types

- enum [Kind](#) { [Kind::Null](#), [Kind::File](#), [Kind::Syslog](#) }

## Public Member Functions

- [Logsheet](#) ()  
Create a [Logsheet](#) that has no backing store. A log entry is maintained, but cannot be permanently stored. This is the Null [Logsheet](#).
- virtual [~Logsheet](#) ()
- void [newEntry](#) ()  
Start a new entry, causing the existing entry to be closed and written.
- std::string [getCurrentEntry](#) () const  
Obtain the contents of the current entry currently under construction.
- void [resetCurrentEntry](#) ()
- uint32\_t [getCurrentEntryNumber](#) () const  
Obtain the current entry number.
- virtual void [write](#) (const std::string &entry)  
Write a string as an entry to the backing store.
- virtual void [writeComment](#) (const std::string &entry)  
Write a string as a comment to the backing store.
- virtual void [writeDebug](#) (const std::string &entry)  
Write a string as a debug entry to the backing store.
- void [setCommit](#) (const bool state)  
Enable or disable the commitment of normal entries to the backing log storage.
- bool [getCommit](#) () const  
Get the current entry commit state.
- void [setDebugCommit](#) (const bool state)  
Enable or disable the commitment of debug entries to the backing log storage.
- bool [getDebugCommit](#) () const  
Get the current debug entry commit state.
- void [setCommentCommit](#) (const bool state)  
Enable or disable the commitment of comment entries to the backing log storage.
- bool [getCommentCommit](#) () const  
Get the current comment entry commit state.
- virtual void [sync](#) ()  
Synchronize any buffered data to the underlying backing store.
- void [setAutoSync](#) (bool state)
- bool [getAutoSync](#) () const

## Static Public Member Functions

- static [Logsheet::Kind](#) [getTypeFromURL](#) (const std::string &url)  
Map the URL scheme, taken from a string containing the entire URL, into a [Logsheet](#) type.
- static bool [lineIsEntry](#) (const std::string &line)  
Helper function to determine whether a string is a valid log entry.
- static bool [lineIsComment](#) (const std::string &line)  
Helper function to determine whether a string is a valid comment log entry.
- static bool [lineIsDebug](#) (const std::string &line)  
Helper function to determine whether a string is a valid debug log entry.
- static std::string [trim](#) (const std::string &entry)  
Trim delimiters from [Logsheet](#) entries.

## Static Public Attributes

- static const char [CommentDelimiter](#) = '#'
- static const char [EntryDelimiter](#) = 'E'
- static const char [DebugDelimiter](#) = 'D'
- static const std::string [DescriptionTag](#)
- static const std::string [FILEURLSCHEME](#)
- static const std::string [SYSLOGURLSCHEME](#)

## Protected Member Functions

- void [incrementEntryNumber](#) ()  
*Increment the current entry number.*
- std::string [getCurrentEntryNumberAsString](#) () const  
*Obtain the current entry 'tag', in 'Edddd' format.*

### F.60.1 Detailed Description

A class to represent a 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 are prefixed with an entry number, which is incremented when the entry is written (either by directly calling [write\(\)](#), or calling [newEntry\(\)](#)).

How the log data is stored is implemented by subclasses of [Logsheet](#).

Note

By default, the entries in the [Logsheet](#) may not be immediately written to the backing store, 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.

### F.60.2 Member Enumeration Documentation

**enum BiometricEvaluation::IO::Logsheet::Kind [strong]**

Enumerator

**Null** No backing store log sheet

**File** File-based log sheet

**Syslog** Syslog daemon backing store

### F.60.3 Constructor & Destructor Documentation

**virtual BiometricEvaluation::IO::Logsheet::~~Logsheet ( ) [virtual]**

Destructor

### F.60.4 Member Function Documentation

**bool BiometricEvaluation::IO::Logsheet::getAutoSync ( ) const**

Return the current auto-sync state.

Returns

true if auto-sync is on, false otherwise.

**bool BiometricEvaluation::IO::Logsheet::getCommentCommit ( ) const**

Get the current comment entry commit state.

Returns

true if comment entries are committed to the backing store, false otherwise.

**bool BiometricEvaluation::IO::Logsheet::getCommit ( ) const**

Get the current entry commit state.

Returns

true if normal entries are to be committed, false if not.

**std::string BiometricEvaluation::IO::Logsheet::getCurrentEntry ( ) const**

Obtain the contents of the current entry currently under construction.

Returns

The text of the current entry.

**uint32\_t BiometricEvaluation::IO::Logsheet::getCurrentEntryNumber ( ) const**

Obtain the current entry number.

Returns

The current entry number.

**std::string BiometricEvaluation::IO::Logsheet::getCurrentEntryNumberAsString ( ) const**  
**[protected]**

Obtain the current entry 'tag', in 'Edddd' format.

Returns

The text of the current entry tag.



**bool BiometricEvaluation::IO::Logsheet::getDebugCommit ( ) const**

Get the current debug entry commit state.

Returns

true if debug entries are committed to the backing store, false otherwise.

**static Logsheet::Kind BiometricEvaluation::IO::Logsheet::getTypeFromURL ( const std::string & url ) [static]**

Map the URL scheme, taken from a string containing the entire URL, into a [Logsheet](#) type.

## Parameters

<i>in</i>	<i>url</i>	The uniform resource locator of the <a href="#">Logsheet</a> .
-----------	------------	--

## Returns

The type of [Logsheet](#) represented by the URL.

## Exceptions

<a href="#">Error::ParameterError</a>	The URL scheme is missing or invalid.
---------------------------------------	---------------------------------------

**static bool BiometricEvaluation::IO::Logsheet::lineIsComment ( const std::string & *line* )**  
**[static]**

Helper function to determine whether a string is a valid comment log entry.

## Parameters

<i>in</i>	<i>line</i>	The string potentially containing a comment entry.
-----------	-------------	--

## Returns

true if the string is a comment entry, false otherwise.

**static bool BiometricEvaluation::IO::Logsheet::lineIsDebug ( const std::string & *line* )** **[static]**

Helper function to determine whether a string is a valid debug log entry.

## Parameters

<i>in</i>	<i>line</i>	The string potentially containing a debug entry.
-----------	-------------	--

## Returns

true if the string is a debug entry, false otherwise.

**static bool BiometricEvaluation::IO::Logsheet::lineIsEntry ( const std::string & *line* )** **[static]**

Helper function to determine whether a string is a valid log entry.

## Parameters

<i>in</i>	<i>line</i>	The string potentially containing a log entry.
-----------	-------------	--

## Returns

true if the string is a log entry, false otherwise.

**void BiometricEvaluation::IO::Logsheet::newEntry ( )**

Start a new entry, causing the existing entry to be closed and written.

Applications do not have to call this method for the first entry, however, as the stream is ready for writing upon construction.

## Exceptions

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

**void BiometricEvaluation::IO::Logsheet::resetCurrentEntry ( )**

Reset the current entry buffer to the beginning.

**void BiometricEvaluation::IO::Logsheet::setAutoSync ( bool state )**

Turn on/off auto-sync of the data. Applications may gain 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 <a href="#">newEntry()</a> is or <a href="#">write()</a> is called. When false, <a href="#">sync()</a> must be called to force a write.
--------------	--

**void BiometricEvaluation::IO::Logsheet::setCommentCommit ( const bool state )**

Enable or disable the commitment of comment entries to the backing log storage.

When comment entry commitment is disabled, calls to writeComment may still be made, but those entries do not appear in the log backing store.

Parameters

<i>in</i>	<i>state</i>	true if comment entries are to be committed, false if not.
-----------	--------------	--

**void BiometricEvaluation::IO::Logsheet::setCommit ( const bool state )**

Enable or disable the commitment of normal entries to the backing log storage.

When entry commitment is disabled, the entry number is not incremented. Entries may be streamed into the object, and new entries created.

Parameters

<i>in</i>	<i>state</i>	True if normal entries are to be committed, false if not.
-----------	--------------	---

**void BiometricEvaluation::IO::Logsheet::setDebugCommit ( const bool state )**

Enable or disable the commitment of debug entries to the backing log storage.

When debug entry commitment is disabled, calls to writeDebug may still be made, but those entries do not appear in the log backing store.

Parameters

<i>in</i>	<i>state</i>	true if debug entries are to be committed, false if not.
-----------	--------------	--

**virtual void BiometricEvaluation::IO::Logsheet::sync ( ) [virtual]**

Synchronize any buffered data to the underlying backing store.

This syncing is dependent on the behavior of the underlying storage mechanism.

Exceptions

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

Reimplemented in [BiometricEvaluation::IO::FileLogsheet](#), and [BiometricEvaluation::IO::SysLogsheet](#).

**static std::string BiometricEvaluation::IO::Logsheet::trim ( const std::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.

**virtual void BiometricEvaluation::IO::Logsheet::write ( const std::string & entry ) [virtual]**

Write a string as an entry to the backing store.

This does not affect the current log entry buffer, but does increment the entry number.

Parameters

in	entry	The text of the log entry.
----	-------	----------------------------

Exceptions

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

Reimplemented in [BiometricEvaluation::IO::FileLogsheet](#), and [BiometricEvaluation::IO::SysLogsheet](#).

**virtual void BiometricEvaluation::IO::Logsheet::writeComment ( const std::string & entry ) [virtual]**

Write a string as a comment to the backing store.

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

in	entry	The text of the comment.
----	-------	--------------------------

Exceptions

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

Reimplemented in [BiometricEvaluation::IO::FileLogsheet](#), and [BiometricEvaluation::IO::SysLogsheet](#).

**virtual void BiometricEvaluation::IO::Logsheet::writeDebug ( const std::string & entry ) [virtual]**

Write a string as a debug entry to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A debug line is prefixed with DebugDelimiter followed by a space.

## Parameters

<code>in</code>	<code>entry</code>	The text of the debug message.
-----------------	--------------------	--------------------------------

## Exceptions

<a href="#"><i><code>Error::StrategyError</code></i></a>	An error occurred when logging.
--	---------------------------------

Reimplemented in [BiometricEvaluation::IO::FileLogsheet](#), and [BiometricEvaluation::IO::SysLogsheet](#).

## F.60.5 Member Data Documentation

**`const char BiometricEvaluation::IO::Logsheet::CommentDelimiter = '#' [static]`**

Delimiter for a comment line in the log sheet.

**`const char BiometricEvaluation::IO::Logsheet::DebugDelimiter = 'D' [static]`**

Delimiter for an debug line in the log sheet.

**`const std::string BiometricEvaluation::IO::Logsheet::DescriptionTag [static]`**

The tag for the description string.

**`const char BiometricEvaluation::IO::Logsheet::EntryDelimiter = 'E' [static]`**

Delimiter for an entry line in the log sheet.

**`const std::string BiometricEvaluation::IO::Logsheet::FILEURLSCHEME [static]`**

The URL scheme to be used for [FileLogsheet](#) URL strings.

**`const std::string BiometricEvaluation::IO::Logsheet::SYSLOGURLSCHEME [static]`**

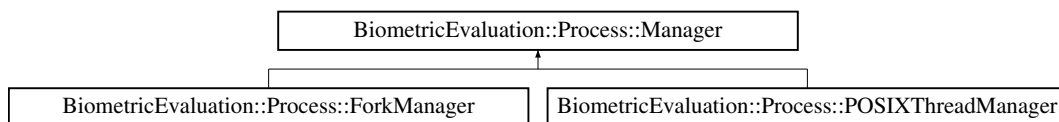
The URL scheme to be used for [SysLogsheet](#) URL strings.

## F.61 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 `std::shared_ptr`  
    < [WorkerController](#) > `addWorker` (`std::shared_ptr`< [Worker](#) > worker)=0

- Adds a [Worker](#) to be managed by this [Manager](#).*

  - virtual uint32\_t [getNumCompletedWorkers](#) () const

*Obtain the number of Workers that have exited.*

  - virtual uint32\_t [getNumActiveWorkers](#) () const

*Obtain the number of Workers that are still working.*

  - virtual uint32\_t [getTotalWorkers](#) () const

*Obtain the number of Workers this class is handling.*

  - virtual void [startWorkers](#) (bool wait=true, bool communicate=false)=0

*Begin [Worker](#)'s work.*

  - virtual void [startWorker](#) (std::shared\_ptr< [WorkerController](#) > worker, bool wait=true, bool communicate=false)=0

*Start a [Worker](#).*

  - virtual void [waitForWorkerExit](#) ()=0

*Block until all Workers have exited.*

  - virtual void [reset](#) ()

*Reuse all Workers.*

  - virtual int32\_t [stopWorker](#) (std::shared\_ptr< [WorkerController](#) > worker)=0

*Ask [Worker](#) to return as soon as possible.*

  - virtual bool [waitForMessage](#) (std::shared\_ptr< [WorkerController](#) > &sender, int \*nextFD=nullptr, int numSeconds=-1) const

*Wait for a message from a [Worker](#).*

  - virtual bool [getNextMessage](#) (std::shared\_ptr< [WorkerController](#) > &sender, [Memory::uint8Array](#) &message, int numSeconds=-1) const

*Obtain a message from a [Worker](#).*

  - virtual void [broadcastMessage](#) ([Memory::uint8Array](#) &message) const

*Send one message to all Workers.*

  - virtual [~Manager](#) ()

*[Manager](#) destructor.*

## Protected Member Functions

- virtual void [\\_wait](#) ()=0
- Do not return until all spawned processes exited.*

## Protected Attributes

- std::vector< std::shared\_ptr< [WorkerController](#) > > [\\_workers](#)
- std::vector< std::shared\_ptr< [WorkerController](#) > > [\\_pendingExit](#)

### F.61.1 Detailed Description

An interface for intranode process management classes.

### F.61.2 Member Function Documentation

**virtual std::shared\_ptr<WorkerController> BiometricEvaluation::Process::Manager::addWorker (**  
**std::shared\_ptr< Worker > *worker* ) [pure virtual]**

Adds a [Worker](#) to be managed by this [Manager](#).

## Parameters

<i>worker</i>	A <a href="#">Worker</a> instance to run.
---------------	---

## Returns

shared\_ptr to worker.

Implemented in [BiometricEvaluation::Process::ForkManager](#), and [BiometricEvaluation::Process::POSI↵XThreadManager](#).

**virtual void BiometricEvaluation::Process::Manager::broadcastMessage ( Memory::uint8Array & message ) const [virtual]**

Send one message to all Workers.

## Parameters

<i>message</i>	The message to send to all Workers.
----------------	-------------------------------------

## Exceptions

<a href="#">Error::StrategyError</a>	Error propagated from the <a href="#">WorkerController</a> .
--------------------------------------	--

**virtual bool BiometricEvaluation::Process::Manager::getNextMessage ( std::shared\_ptr< WorkerController > & sender, Memory::uint8Array & message, int numSeconds = -1 ) const [virtual]**

Obtain a message from a [Worker](#).

## Parameters

out	<i>sender</i>	Reference to a shared pointer of the <a href="#">WorkerController</a> that sent the message.
out	<i>message</i>	Reference to a buffer to hold the message.
in	<i>numSeconds</i>	Number of seconds to wait for a message, or < 0 to block.

## Returns

true if there is a message, false otherwise.

## Exceptions

<a href="#">Error::ObjectDoesNot↵Exist</a>	(Unexpected) widowed pipe.
<a href="#">Error::StrategyError</a>	Error receiving message.

**virtual uint32\_t BiometricEvaluation::Process::Manager::getNumActiveWorkers ( ) const [virtual]**

Obtain the number of Workers that are still working.

## Returns

The number of Workers that are still working.



Exceptions

<a href="#"><i>Error::StrategyError</i></a>	No Workers have started working yet.
---	--------------------------------------

**virtual uint32\_t BiometricEvaluation::Process::Manager::getNumCompletedWorkers ( ) const [virtual]**

Obtain the number of Workers that have exited.

Returns

The number of Workers that have exited.

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	No Workers have started working yet.
---	--------------------------------------

**virtual uint32\_t BiometricEvaluation::Process::Manager::getTotalWorkers ( ) const [virtual]**

Obtain the number of Workers this class is handling.

Returns

Number of Workers.

**virtual void BiometricEvaluation::Process::Manager::reset ( ) [virtual]**

Reuse all Workers.

Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	At least one <a href="#">Worker</a> is still working.
--	---

**virtual void BiometricEvaluation::Process::Manager::startWorker ( std::shared\_ptr< WorkerController > worker, bool wait = true, bool communicate = false ) [pure virtual]**

Start a [Worker](#).

Parameters

	<i>worker</i>	Pointer to a <a href="#">WorkerController</a> that is being managed by this <a href="#">Manager</a> instance.
	<i>wait</i>	Whether or not to wait for this <a href="#">Worker</a> to exit before returning control to the caller.
<i>in</i>	<i>communicate</i>	Whether or not to enable communication among the Workers and Managers.

Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	worker is already working.
<a href="#"><i>Error::StrategyError</i></a>	worker is not managed by this <a href="#">Manager</a> 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::ForkManager](#), and [BiometricEvaluation::Process::POSIXThreadManager](#).

---

```
virtual void BiometricEvaluation::Process::Manager::startWorkers ( bool wait = true, bool  
communicate = false ) [pure virtual]
```

Begin [Worker](#)'s work.

## Parameters

in	<i>wait</i>	Whether or not to wait for all Workers to return before returning.
in	<i>communicate</i>	Whether or not to enable communication among the Workers and Managers.

## Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	At least one <a href="#">Worker</a> is already working.
<a href="#"><i>Error::StrategyError</i></a>	Problem starting Workers.

Implemented in [BiometricEvaluation::Process::ForkManager](#), and [BiometricEvaluation::Process::POSI↔XThreadManager](#).

**virtual int32\_t BiometricEvaluation::Process::Manager::stopWorker ( std::shared\_ptr< WorkerController > worker ) [pure virtual]**

Ask [Worker](#) to return as soon as possible.

## Parameters

<i>worker</i>	Pointer to the <a href="#">WorkerController</a> that should be stopped.
---------------	---

## Returns

Return code of worker.

## Exceptions

<a href="#"><i>Error::ObjectDoesNot↔Exist</i></a>	worker is not working.
<a href="#"><i>Error::StrategyError</i></a>	Problem asking worker to stop.

Implemented in [BiometricEvaluation::Process::ForkManager](#), and [BiometricEvaluation::Process::POSI↔XThreadManager](#).

**virtual bool BiometricEvaluation::Process::Manager::waitForMessage ( std::shared\_ptr< WorkerController > & sender, int \* nextFD = nullptr, int numSeconds = -1 ) const [virtual]**

Wait for a message from a [Worker](#).

## Parameters

out	<i>sender</i>	Reference to a shared pointer of the <a href="#">WorkerController</a> that sent the message.
in, out	<i>nextFD</i>	Location to store a pipe that has data to read.
in	<i>numSeconds</i>	Number of seconds to wait for a message, or < 0 to block.

## Returns

true if there is a [Worker](#) sending a message false otherwise or if an error occurred.

**virtual void BiometricEvaluation::Process::Manager::waitForWorkerExit ( ) [pure virtual]**

Block until all Workers have exited.

Use this method if wait=false was set during a call to startWorker(s) but now wait=true is desired.

Implemented in [BiometricEvaluation::Process::ForkManager](#), and [BiometricEvaluation::Process::POSI↔XThreadManager](#).

### F.61.3 Member Data Documentation

**std::vector<std::shared\_ptr<WorkerController> > BiometricEvaluation::Process::Manager::pendingExit** [protected]

Workers that are about to exit (stop requested).

**std::vector<std::shared\_ptr<WorkerController> > BiometricEvaluation::Process::Manager::workers** [protected]

Workers that have been added.

## F.62 BiometricEvaluation::IO::ManifestEntry Struct Reference

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

### Public Attributes

- long [offset](#)
- uint64\_t [size](#)

### F.62.1 Detailed Description

Info about a single archive element

### F.62.2 Member Data Documentation

**long BiometricEvaluation::IO::ManifestEntry::offset**

The offset from the beginning of the file/memory

**uint64\_t BiometricEvaluation::IO::ManifestEntry::size**

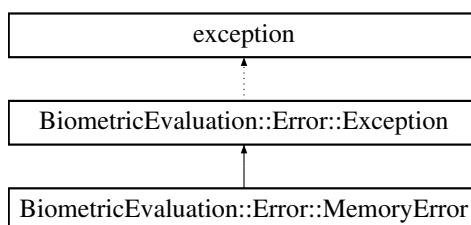
The length from offset this element spans

## F.63 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](#) (std::string info)

### F.63.1 Detailed Description

An error occurred when allocating an object.

### F.63.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::MemoryError::MemoryError ( )**

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

**BiometricEvaluation::Error::MemoryError::MemoryError ( std::string info )**

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

## F.64 BiometricEvaluation::Process::MessageCenter Class Reference

```
#include <be_process_messagecenter.h>
```

## Public Member Functions

- [MessageCenter](#) (uint32\_t port=[MessageCenter::DEFAULT\\_PORT](#))  
*Constructor.*
- bool [hasUnseenMessages](#) () const  
*Determine whether or not there are unseen messages.*
- bool [getNextMessage](#) (uint32\_t &clientID, [Memory::uint8Array](#) &message, int numSeconds=-1)  
*Get the next available message.*
- void [sendResponse](#) (uint32\_t clientID, const [Memory::uint8Array](#) &message) const  
*Send a message to a client.*
- void [disconnectClient](#) (uint32\_t clientID)  
*Break the connection with a client.*

## Static Public Attributes

- static const int [CONNECTION\\_BACKLOG](#) = 10
- static const uint16\_t [DEFAULT\\_PORT](#) = 7899
- static const int [DEFAULT\\_TIMEOUT](#) = 1
- static const uint64\_t [MAX\\_MESSAGE\\_LENGTH](#) = 255

### F.64.1 Detailed Description

Convenience for asynchronous TCP socket message passing.

### F.64.2 Constructor & Destructor Documentation

**BiometricEvaluation::Process::MessageCenter::MessageCenter ( uint32\_t *port* = MessageCenter::DEFAULT\_PORT )**

Constructor.

Parameters

<i>port</i>	Listening port.
-------------	-----------------

### F.64.3 Member Function Documentation

**void BiometricEvaluation::Process::MessageCenter::disconnectClient ( uint32\_t *clientID* )**

Break the connection with a client.

Parameters

<i>clientID</i>	ID of the client to disconnect.
-----------------	---------------------------------

**bool BiometricEvaluation::Process::MessageCenter::getNextMessage ( uint32\_t & *clientID*,  
Memory::uint8Array & *message*, int *numSeconds* = -1 )**

Get the next available message.

Parameters

out	<i>clientID</i>	ID of the client that sent the message.
in, out	<i>message</i>	Message received.
in	<i>numSeconds</i>	Number of seconds to wait for a message, or < 0 to block indefinitely.

Returns

true if a message was received before timing out.

**bool BiometricEvaluation::Process::MessageCenter::hasUnseenMessages ( ) const**

Determine whether or not there are unseen messages.

Returns

true if a message has been received and not read.

Note

Returns immediately.

**void BiometricEvaluation::Process::MessageCenter::sendResponse ( uint32\_t *clientID*, const  
Memory::uint8Array & *message* ) const**

Send a message to a client.

Parameters

<i>clientID</i>	ID of client to receive message.
<i>message</i>	Message to send client.

### F.64.4 Member Data Documentation

**const int BiometricEvaluation::Process::MessageCenter::CONNECTION\_BACKLOG = 10  
[static]**

Number of outstanding connections.

**const uint16\_t BiometricEvaluation::Process::MessageCenter::DEFAULT\_PORT = 7899 [static]**

Default port used for messages.

**const int BiometricEvaluation::Process::MessageCenter::DEFAULT\_TIMEOUT = 1 [static]**

Default number of seconds to wait between polls.

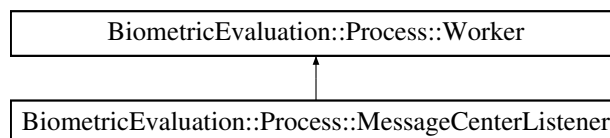
**const uint64\_t BiometricEvaluation::Process::MessageCenter::MAX\_MESSAGE\_LENGTH = 255 [static]**

Maximum length of a message.

## F.65 BiometricEvaluation::Process::MessageCenterListener Class Reference

```
#include <be_process_mclistener.h>
```

Inheritance diagram for BiometricEvaluation::Process::MessageCenterListener:



### Public Member Functions

- int32\_t [workerMain \(\)](#)

*The method that will get called to start execution by a ProcessManager.*

### Static Public Attributes

- static const std::string [PARAM\\_PORT](#)

### Additional Inherited Members

#### F.65.1 Detailed Description

Accepts new connections and spawns message receivers.

#### F.65.2 Member Function Documentation

**int32\_t BiometricEvaluation::Process::MessageCenterListener::workerMain ( ) [virtual]**

The method that will get called 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.

Implements [BiometricEvaluation::Process::Worker](#).

### F.65.3 Member Data Documentation

**const std::string BiometricEvaluation::Process::MessageCenterListener::PARAM\_PORT [static]**

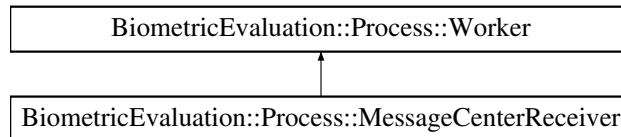
Parameter used to pass port number

## F.66 BiometricEvaluation::Process::MessageCenterReceiver Class Reference

Receives message from a client, forwarding to the central [MessageCenter](#).

```
#include <be_process_mcreceiver.h>
```

Inheritance diagram for BiometricEvaluation::Process::MessageCenterReceiver:



### Public Member Functions

- int32\_t [workerMain\(\)](#)
- [MessageCenterReceiver\(\)](#)=default
- [~MessageCenterReceiver\(\)](#)=default

### Static Public Attributes

- static const std::string [PARAM\\_CLIENT\\_SOCKET](#)
- static const std::string [PARAM\\_CLIENT\\_ID](#)
- static const std::string [MSG\\_DISCONNECT](#)

### Additional Inherited Members

#### F.66.1 Detailed Description

Receives message from a client, forwarding to the central [MessageCenter](#).

#### F.66.2 Constructor & Destructor Documentation

**BiometricEvaluation::Process::MessageCenterReceiver::MessageCenterReceiver ( ) [default]**

Default constructor.

**BiometricEvaluation::Process::MessageCenterReceiver::~~MessageCenterReceiver ( ) [default]**

Default destructor.

### F.66.3 Member Function Documentation

**int32\_t BiometricEvaluation::Process::MessageCenterReceiver::workerMain ( ) [virtual]**

Receive loop.

Implements [BiometricEvaluation::Process::Worker](#).

### F.66.4 Member Data Documentation

**const std::string BiometricEvaluation::Process::MessageCenterReceiver::MSG\_DISCONNECT [static]**

Message sent when client should disconnect.

**const std::string BiometricEvaluation::Process::MessageCenterReceiver::PARAM\_CLIENT\_ID [static]**

Parameter used to pass an ID to the client.

**const std::string BiometricEvaluation::Process::MessageCenterReceiver::PARAM\_CLIENT\_SOCKET [static]**

Parameter used to pass client socket FD.

## F.67 BiometricEvaluation::MPI::MessageTag Class Reference

The types of messages sent between [MPI](#) task processes.

```
#include <be_mpi.h>
```

### Public Types

- enum [Kind](#) { **Control** = 0, **Data** = 1, **OOB** = 2 }

### F.67.1 Detailed Description

The types of messages sent between [MPI](#) task processes.

### F.67.2 Member Enumeration Documentation

**enum BiometricEvaluation::MPI::MessageTag::Kind**

Enumerator

**Data** A control message (start, exit, etc.

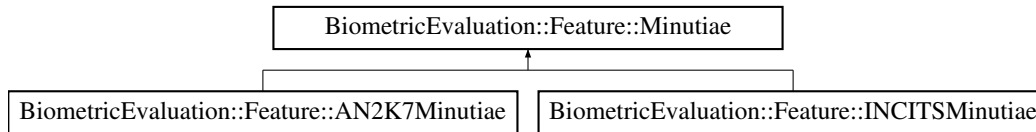
**OOB** A data message.

## F.68 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](#) [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.*

### F.68.1 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.

## F.69 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](#) **type**
- [Image::Coordinate](#) **coordinate**
- unsigned int **theta**
- bool **has\_quality**
- unsigned int **quality**

### F.69.1 Detailed Description

Representation of a finger minutiae data point.

## F.70 BiometricEvaluation::Feature::MPEGFacePoint Struct Reference

Representation of a feature point and a set of points.

```
#include <be_feature_mpegfacepoint.h>
```

### Public Attributes

- `uint8_t` **type**
- `uint8_t` **major**
- `uint8_t` **minor**
- [BiometricEvaluation::Image::Coordinate](#) **coordinate**

### F.70.1 Detailed Description

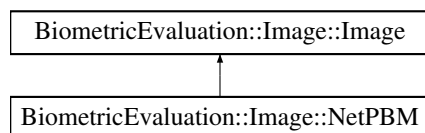
Representation of a feature point and a set of points.

## F.71 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, **Binary**↵  
**PortableBitmap** = 4,  
**BinaryPortableGraymap** = 5, **BinaryPortablePixmap** = 6 }

### Public Member Functions

- **NetPBM** (const `uint8_t` \*data, const `uint64_t` size)
- [Memory::uint8Array](#) **getRawData** () const  
*Accessor for the raw image data. The data returned should not be compressed or encoded.*
- [Memory::uint8Array](#) **getRawGrayscaleData** (`uint8_t` depth=8) const  
*Accessor for decompressed data in grayscale.*

## Static Public Member Functions

- static bool [isNetPBM](#) (const uint8\_t \*data, uint64\_t size)
- static void [skipLine](#) (const uint8\_t \*data, size\_t dataSize, size\_t &offset)  
*Skip an entire line of input, placing offset at the first character after the newline.*
- static void [skipComment](#) (const uint8\_t \*data, size\_t dataSize, size\_t &offset)  
*Skip a block of comments in input.*
- static std::string [getNextValue](#) (const uint8\_t \*data, size\_t dataSize, size\_t &offset, size\_t sizeOfValue=0)  
*Obtain the next space-separated value from data, beginning at offset.*
- static [Memory::uint8Array](#) [ASCIIBitmapTo8Bit](#) (const uint8\_t \*bitmap, uint64\_t bitmapSize, uint32\_t width, uint32\_t height)  
*Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.*
- static [Memory::uint8Array](#) [ASCIIPixmapToBinaryPixmap](#) (const uint8\_t \*ASCIIBuf, uint64\_t ASCII↵BufSize, uint32\_t width, uint32\_t height, uint8\_t depth, uint32\_t maxColor)  
*Convert an ASCII pixel map buffer into a binary pixel map buffer.*
- static [Memory::uint8Array](#) [BinaryBitmapTo8Bit](#) (const uint8\_t \*bitmap, uint64\_t bitmapSize, uint32\_t width, uint32\_t height)  
*Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.*

## Additional Inherited Members

### F.71.1 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.

### F.71.2 Member Function Documentation

**static [Memory::uint8Array](#) [BiometricEvaluation::Image::NetPBM::ASCIIBitmapTo8Bit](#) ( const uint8\_t \* *bitmap*, uint64\_t *bitmapSize*, uint32\_t *width*, uint32\_t *height* ) [static]**

Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Parameters

<i>bitmap</i>	Bitmap data buffer.
<i>bitmapSize</i>	<a href="#">Size</a> of bitmap.
<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>	<a href="#">Error</a> extracting a value from the bitmap.
---------------------	---

**static Memory::uint8Array BiometricEvaluation::Image::NetPBM::ASCIIPixmapToBinaryPixmap ( const uint8\_t \* *ASCIIBuf*, uint64\_t *ASCIIBufSize*, uint32\_t *width*, uint32\_t *height*, uint8\_t *depth*, uint32\_t *maxColor* ) [static]**

Convert an ASCII pixel map buffer into a binary pixel map buffer.

Parameters

<i>ASCIIBuf</i>	ASCII pixel map data buffer.
<i>ASCIIBufSize</i>	<a href="#">Size</a> of <i>ASCIIBuf</i> .
<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>	<a href="#">Error</a> extracting a value from the pixel map.
<a href="#">Error::ParameterError</a>	Invalid value for depth, must be a multiple of <a href="#">Image::bitsPerComponent</a> .

**static Memory::uint8Array BiometricEvaluation::Image::NetPBM::BinaryBitmapTo8Bit ( const uint8\_t \* *bitmap*, uint64\_t *bitmapSize*, uint32\_t *width*, uint32\_t *height* ) [static]**

Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Parameters

<i>bitmap</i>	Bitmap data buffer.
<i>bitmapSize</i>	<a href="#">Size</a> of <i>bitmap</i> .
<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>	<a href="#">Error</a> extracting a value from the bitmap.
---------------------	---

**static std::string BiometricEvaluation::Image::NetPBM::getNextValue ( const uint8\_t \* *data*, size\_t *dataSize*, 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>dataSize</i>	<a href="#">Size</a> of data.
<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 <i>sizeOfValue</i> length. 0 assumes space-separated.

## Returns

Next value from data.

**Memory::uint8Array BiometricEvaluation::Image::NetPBM::getRawData ( ) const [virtual]**

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

## Returns

AutoArray holding raw image data.

## Exceptions

<a href="#">Error::DataError</a>	<a href="#">Error</a> decompressing image data.
<a href="#">Error::NotImplemented</a>	Compression type not supported.

## 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](#).

**Memory::uint8Array BiometricEvaluation::Image::NetPBM::getRawGrayscaleData ( uint8\_t depth = 8 ) const [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

AutoArray holding raw grayscale image data.

## Exceptions

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

## Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. *depth* adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

```
static bool BiometricEvaluation::Image::NetPBM::isNetPBM ( const uint8_t * data, uint64_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 data.

## Returns

true if data appears to be a netpbm image, false otherwise.

**static void BiometricEvaluation::Image::NetPBM::skipComment ( const uint8\_t \* *data*, size\_t *dataSize*, size\_t & *offset* ) [static]**

Skip a block of comments in input.

## Parameters

<i>data</i>	Buffer with comment to be skipped.
<i>dataSize</i>	<a href="#">Size</a> of data
<i>offset</i>	Position within data from which the rest of the line should be read.

## Exceptions

<i>out_of_range</i>	End of line not encountered before end of data or on last line of data.
---------------------	---

**static void BiometricEvaluation::Image::NetPBM::skipLine ( const uint8\_t \* *data*, size\_t *dataSize*, size\_t & *offset* ) [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>dataSize</i>	<a href="#">Size</a> of data.
<i>offset</i>	Position within data from which the rest of the line should be read.

## Exceptions

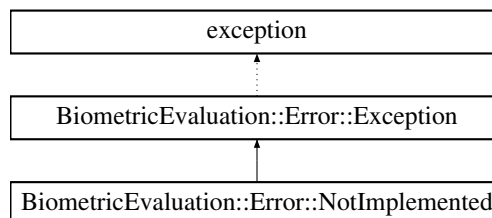
<i>out_of_range</i>	End of line not encountered before end of data or on last line of data.
---------------------	---

## F.72 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](#) (std::string info)

### F.72.1 Detailed Description

A [NotImplemented](#) object is thrown when the underlying implementation of this interface has not or could not be created.

### F.72.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::NotImplemented::NotImplemented ( )**

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

**BiometricEvaluation::Error::NotImplemented::NotImplemented ( std::string *info* )**

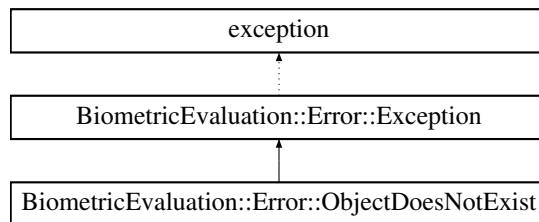
Construct a [NotImplemented](#) object with an information string appended to the default information string.

## F.73 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](#) (std::string info)

### F.73.1 Detailed Description

The named object does not exist.

### F.73.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( )**

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

**BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( std::string *info* )**

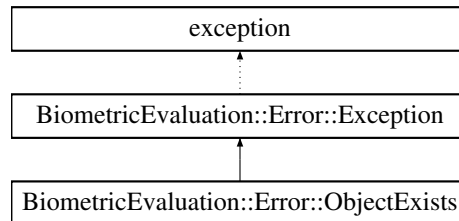
Construct a [ObjectDoesNotExist](#) object with an information string appended to the default information string.

## F.74 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](#) (std::string info)

#### F.74.1 Detailed Description

The named object exists and will not be replaced.

#### F.74.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::ObjectExists::ObjectExists ( )**

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

**BiometricEvaluation::Error::ObjectExists::ObjectExists ( std::string *info* )**

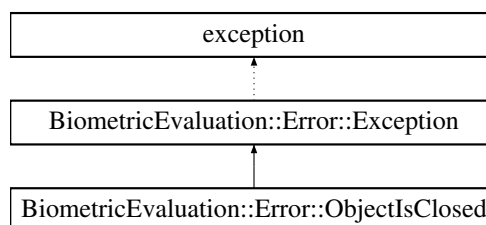
Construct a [ObjectExists](#) object with an information string appended to the default information string.

## F.75 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](#) (std::string info)

### F.75.1 Detailed Description

The object is closed.

### F.75.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ( )**

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

**BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ( std::string *info* )**

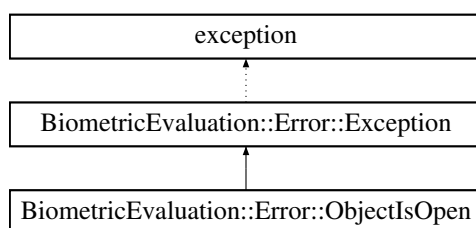
Construct a [ObjectIsClosed](#) object with an information string appended to the default information string.

## F.76 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](#) (std::string info)

### F.76.1 Detailed Description

The object is already opened.

### F.76.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ( )**

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

**BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ( std::string *info* )**

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

## F.77 BiometricEvaluation::Memory::OrderedMap< Key, T > Class Template Reference

```
#include <be_memory_orderedmap.h>
```

### Public Types

- using **container** = typename std::unordered\_map< Key, T >
- using **iterator** = [OrderedMapIterator](#)< Key, T >
- using **const\_iterator** = [OrderedMapConstIterator](#)< Key, T >
- using **size\_type** = typename container::size\_type
- using **value\_type** = typename container::value\_type
- using **key\_type** = Key
- using **mapped\_type** = T
- using **key\_equal** = typename container::key\_equal

### Public Member Functions

- [OrderedMap](#) ()
- bool [push\\_back](#) (const value\_type &value)  
*Insert an element at the end of the collection.*
- void [erase](#) (iterator pos)  
*Remove an element from the collection.*
- void [erase](#) (const Key &key)  
*Remove an element from the collection.*
- [iterator](#) [begin](#) ()
- [const\\_iterator](#) [begin](#) () const
- [const\\_iterator](#) [cbegin](#) () const
- [iterator](#) [end](#) ()
- [const\\_iterator](#) [end](#) () const
- [const\\_iterator](#) [cend](#) () const
- size\_type [size](#) () const
- bool [keyExists](#) (const Key &key) const  
*Determine if a value exists in the container.*
- const [OrderedMapIterator](#)< Key, T > [find](#) (const Key &key) const  
*Obtain an iterator to a particular key.*
- std::shared\_ptr< value\_type > [find\\_quick](#) (const Key &key) const
- T & [operator](#)[ ] (const Key &key)  
*Subscripting operator.*
- key\_equal [key\\_eq](#) () const
- [~OrderedMap](#) ()

### Friends

- class [OrderedMapIterator](#)< Key, T >
- class [OrderedMapConstIterator](#)< Key, T >

### F.77.1 Detailed Description

**template<class Key, class T>class BiometricEvaluation::Memory::OrderedMap< Key, T >**

A map where insertion order is preserved and elements are unique.

### F.77.2 Constructor & Destructor Documentation

**template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::OrderedMap ( )**

Constructor.

**template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::~~OrderedMap ( )**

Destructor

### F.77.3 Member Function Documentation

**template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::begin ( )**

Returns

Iterator at the first element of the collection.

**template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const\_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::begin ( ) const**

Returns

Iterator at the first element of the collection.

**template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const\_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::cbegin ( ) const**

Returns

Iterator at the first element of the collection.

**template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const\_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::cend ( ) const**

Returns

Iterator beyond the last element of the collection.

**template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::end ( )**

Returns

Iterator beyond the last element of the collection.

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator  
BiometricEvaluation::Memory::OrderedMap< Key, T >::end ( ) const
```

Returns

Iterator beyond the last element of the collection.

```
template<class Key , class T > void BiometricEvaluation::Memory::OrderedMap< Key, T >::erase (  
iterator pos )
```

Remove an element from the collection.

Parameters

<i>pos</i>	Iterator to element at the position which should be removed.
------------	--

Note

Complexity: Average case: O(1), worst case O(size()).

```
template<class Key, class T > void BiometricEvaluation::Memory::OrderedMap< Key, T >::erase (  
const Key & key )
```

Remove an element from the collection.

Parameters

<i>key</i>	Key of the element to remove.
------------	-------------------------------

```
template<class Key, class T > const BiometricEvaluation::Memory::OrderedMapIterator< Key, T >  
BiometricEvaluation::Memory::OrderedMap< Key, T >::find ( const Key & key ) const
```

Obtain an iterator to a particular key.

Note

Complexity is O(n).

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::key_equal  
BiometricEvaluation::Memory::OrderedMap< Key, T >::key_eq ( ) const
```

Returns

Function that compares keys for equality.

```
template<class Key, class T > bool BiometricEvaluation::Memory::OrderedMap< Key, T >::keyExists  
( const Key & key ) const
```

Determine if a value exists in the container.

Parameters

<i>key</i>	Key to search the container for.
------------	----------------------------------

## Returns

Whether or not key exists in this container.

## Note

Complexity is O(1).

```
template<class Key, class T > T & BiometricEvaluation::Memory::OrderedMap< Key, T
>::operator[] ( const Key & key )
```

Subscripting operator.

Parameters

<i>key</i>	Key used to index into the map.
------------	---------------------------------

## Returns

Value for key, which may be a new value.

```
template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMap< Key, T
>::push_back ( const value.type & value )
```

Insert an element at the end of the collection.

Parameters

<i>value</i>	Value to insert.
--------------	------------------

## Returns

Whether or not the object was inserted.

## Note

Complexity: Average case: O(1), worst case O(size()).

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::size_type
BiometricEvaluation::Memory::OrderedMap< Key, T >::size ( ) const
```

## Returns

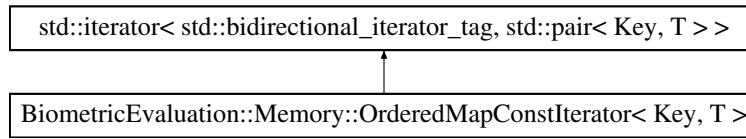
Number of elements in the collection.

## F.78 BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > Class Template Reference

```
#include <be_memory_orderedmap.h>
```

Inheritance diagram for BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >:





## Public Types

- using **reference** = typename std::iterator\_traits< [OrderedMapConstIterator](#) >::reference
- using **const\_reference** = const typename std::iterator\_traits< [OrderedMapConstIterator](#) >::reference
- using **pointer** = typename std::iterator\_traits< [OrderedMapConstIterator](#) >::pointer
- using **const\_pointer** = const typename std::iterator\_traits< [OrderedMapConstIterator](#) >::pointer
- using **value\_type** = typename std::iterator\_traits< [OrderedMapConstIterator](#) >::value\_type
- using **difference\_type** = typename std::iterator\_traits< [OrderedMapConstIterator](#) >::difference\_type

## Public Member Functions

- [OrderedMapConstIterator](#) ()
- [OrderedMapConstIterator](#) (const [OrderedMapIterator](#)< Key, T > &iterator)
- [~OrderedMapConstIterator](#) ()
- const\_reference [operator\\*](#) () const
- const\_pointer [operator->](#) () const
- [OrderedMapConstIterator](#) & [operator++](#) ()
- [OrderedMapConstIterator](#) & [operator++](#) (int dummy)
- [OrderedMapConstIterator](#) & [operator--](#) ()
- [OrderedMapConstIterator](#) & [operator--](#) (int dummy)
- bool [operator==](#) (const [OrderedMapConstIterator](#) &rhs) const  
*Test for iterator equality.*
- bool [operator!=](#) (const [OrderedMapConstIterator](#) &rhs) const  
*Test for iterator equality.*

## Friends

- class [OrderedMap](#)< Key, T >

### F.78.1 Detailed Description

template<class Key, class T>class BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >

Const Iterator for OrderedMaps.

### F.78.2 Constructor & Destructor Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::OrderedMapConstIterator ( )

Constructor

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T
>::OrderedMapConstIterator ( const OrderedMapIterator< Key, T > & iterator )
```

Iterator to ConstIterator converter

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T
>::~~OrderedMapConstIterator ( )
```

Destructor

### F.78.3 Member Function Documentation

```
template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapConstIterator< Key,
T >::operator!= ( const OrderedMapConstIterator< Key, T > & rhs ) const
```

Test for iterator equality.

Parameters

<i>rhs</i>	Object on the right-hand side of the expression.
------------	--

Returns

Whether or not this iterator is not equivalent to rhs.

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T
>::const_reference BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator* (
) const
```

Returns

Reference to the current iterated pair.

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >
& BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator++ ( )
```

Move to the next pair

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >
& BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator++ ( int dummy )
```

Move to the next pair

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >
& BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator-- ( )
```

Move to the previous pair.

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >
& BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator-- ( int dummy )
```

Move to the previous pair.

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T
>::const_pointer BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator-> (
) const
```

Returns

Pointer to the current iterated pair.

```
template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapConstIterator< Key,
T >::operator==( const OrderedMapConstIterator< Key, T > & rhs ) const
```

Test for iterator equality.

Parameters

<i>rhs</i>	Object on the right-hand side of the expression.
------------	--

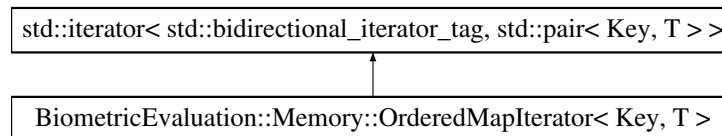
Returns

Whether or not this iterator is equivalent to rhs.

## F.79 BiometricEvaluation::Memory::OrderedMapIterator< Key, T > Class Template Reference

```
#include <be_memory_orderedmap.h>
```

Inheritance diagram for BiometricEvaluation::Memory::OrderedMapIterator< Key, T >:



### Public Types

- using **reference** = typename std::iterator\_traits< [OrderedMapIterator](#) >::reference
- using **pointer** = typename std::iterator\_traits< [OrderedMapIterator](#) >::pointer
- using **value\_type** = typename std::iterator\_traits< [OrderedMapIterator](#) >::value\_type
- using **difference\_type** = typename std::iterator\_traits< [OrderedMapIterator](#) >::difference\_type

### Public Member Functions

- [OrderedMapIterator](#) ()
- [~OrderedMapIterator](#) ()
- reference [operator\\*](#) () const
- pointer [operator->](#) () const
- [OrderedMapIterator](#) & [operator++](#) ()
- [OrderedMapIterator](#) & [operator++](#) (int dummy)
- [OrderedMapIterator](#) & [operator--](#) ()
- [OrderedMapIterator](#) & [operator--](#) (int dummy)
- bool [operator==](#) (const [OrderedMapIterator](#) &rhs) const

*Test for iterator equality.*

- `bool operator!=(const OrderedMapIterator &rhs) const`

*Test for iterator equality.*

## Friends

- `class OrderedMap< Key, T >`
- `class OrderedMapConstIterator< Key, T >`

## F.79.1 Detailed Description

`template<class Key, class T>class BiometricEvaluation::Memory::OrderedMapIterator< Key, T >`

Iterator for OrderedMaps.

## F.79.2 Constructor & Destructor Documentation

`template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::OrderedMapIterator ( )`

Constructor

`template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::~~OrderedMapIterator ( )`

Destructor

## F.79.3 Member Function Documentation

`template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator!=( const OrderedMapIterator< Key, T > & rhs ) const`

Test for iterator equality.

Parameters

<i>rhs</i>	Object on the right-hand side of the expression.
------------	--

Returns

Whether or not this iterator is not equivalent to rhs.

`template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::reference BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator* ( ) const`

Returns

Reference to the current iterated pair.

`template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator++ ( )`

Move to the next pair

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > &
BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator++ ( int dummy )
```

Move to the next pair

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > &
BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator-- ( )
```

Move to the previous pair.

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > &
BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator-- ( int dummy )
```

Move to the previous pair.

```
template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T
>::pointer BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator-> ( ) const
```

Returns

Pointer to the current iterated pair.

```
template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapIterator< Key, T
>::operator== ( const OrderedMapIterator< Key, T > & rhs ) const
```

Test for iterator equality.

Parameters

<i>rhs</i>	Object on the right-hand side of the expression.
------------	--

Returns

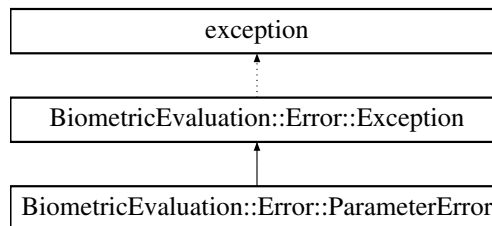
Whether or not this iterator is equivalent to rhs.

## F.80 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](#) (std::string info)

### F.80.1 Detailed Description

An invalid parameter was passed to a constructor or method.

### F.80.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::ParameterError::ParameterError** ( )

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

**BiometricEvaluation::Error::ParameterError::ParameterError** ( `std::string info` )

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

## F.81 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification Class Reference

Pattern classification codes.

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

### Classes

- struct [Entry](#)

### Public Types

- using **Entry** = struct [Entry](#)

### F.81.1 Detailed Description

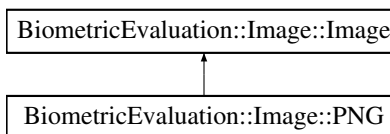
Pattern classification codes.

## F.82 BiometricEvaluation::Image::PNG Class Reference

A PNG-encoded image.

```
#include <be_image_png.h>
```

Inheritance diagram for BiometricEvaluation::Image::PNG:



### Public Member Functions

- **PNG** (const uint8\_t \*data, const uint64\_t size)
- [Memory::uint8Array getRawData](#) () const  
*Accessor for the raw image data. The data returned should not be compressed or encoded.*
- [Memory::uint8Array getRawGrayscaleData](#) (uint8\_t depth=8) const

*Accessor for decompressed data in grayscale.*

## Static Public Member Functions

- static bool [isPNG](#) (const uint8\_t \*data, uint64\_t size)

## Additional Inherited Members

### F.82.1 Detailed Description

A PNG-encoded image.

### F.82.2 Member Function Documentation

**Memory::uint8Array BiometricEvaluation::Image::PNG::getRawData ( ) const [virtual]**

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

Returns

AutoArray holding raw image data.

Exceptions

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

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

**Memory::uint8Array BiometricEvaluation::Image::PNG::getRawGrayscaleData ( uint8\_t depth = 8 ) const [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

AutoArray holding raw grayscale image data.

Exceptions

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

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

**static bool BiometricEvaluation::Image::PNG::isPNG ( const uint8\_t \* data, uint64\_t size ) [static]**

Whether or not data is a [PNG](#) 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 [PNG](#) image, false otherwise

## F.83 BiometricEvaluation::Face::PoseAngle Struct Reference

Representation of pose angle and uncertainty.

```
#include <be_face.h>
```

### Public Attributes

- uint8\_t **yaw**
- uint8\_t **pitch**
- uint8\_t **roll**
- uint8\_t **yawUncertainty**
- uint8\_t **pitchUncertainty**
- uint8\_t **rollUncertainty**

### F.83.1 Detailed Description

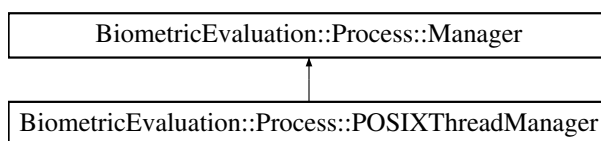
Representation of pose angle and uncertainty.

## F.84 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](#) ()
- std::shared\_ptr< [WorkerController](#) > **addWorker** (std::shared\_ptr< [Worker](#) > worker)  
*Adds a [Worker](#) to be managed by this [Manager](#).*
- void **startWorkers** (bool wait=true, bool communicate=false)  
*Begin [Worker](#)'s work.*
- void **startWorker** (std::shared\_ptr< [WorkerController](#) > worker, bool wait=true, bool communicate=false)  
*Start a [Worker](#).*



- `int32_t stopWorker (std::shared_ptr< WorkerController > workerController)`  
Ask *Worker* to exit.
- `void waitForWorkerExit ()`  
Block until all *Workers* have exited.
- `~POSIXThreadManager ()`  
*~POSIXThreadManager* destructor.

## Additional Inherited Members

### F.84.1 Detailed Description

*Manager* implementation that starts *Workers* in POSIX threads.

### F.84.2 Constructor & Destructor Documentation

**BiometricEvaluation::Process::POSIXThreadManager::POSIXThreadManager ( )**

*POSIXThreadManager* constructor.

### F.84.3 Member Function Documentation

**std::shared\_ptr<WorkerController> BiometricEvaluation::Process::POSIXThreadManager::addWorker ( std::shared\_ptr< Worker > worker ) [virtual]**

Adds a *Worker* to be managed by this *Manager*.

Parameters

<i>worker</i>	A <i>Worker</i> instance to run.
---------------	----------------------------------

Returns

`shared_ptr` to *worker*.

Implements *BiometricEvaluation::Process::Manager*.

**void BiometricEvaluation::Process::POSIXThreadManager::startWorker ( std::shared\_ptr< WorkerController > worker, bool wait = true, bool communicate = false ) [virtual]**

Start a *Worker*.

Parameters

<i>worker</i>	Pointer to a <i>WorkerController</i> that is being managed by this <i>Manager</i> instance.
<i>wait</i>	Whether or not to wait for this <i>Worker</i> to exit before returning control to the caller.
<i>communicate</i>	Whether or not to enable communication among the <i>Workers</i> and <i>Managers</i> .

Exceptions

<i>Error::ObjectExists</i>	<i>worker</i> is already working.
<i>Error::StrategyError</i>	<i>worker</i> is not managed by this <i>Manager</i> instance.

Implements *BiometricEvaluation::Process::Manager*.

**void BiometricEvaluation::Process::POSIXThreadManager::startWorkers ( bool wait = true, bool communicate = false ) [virtual]**

Begin *Worker*'s work.

## Parameters

in	<i>wait</i>	Whether or not to wait for all Workers to return before returning.
in	<i>communicate</i>	Whether or not to enable communication among the Workers and Managers.

## Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	At least one <a href="#">Worker</a> is already working.
<a href="#"><i>Error::StrategyError</i></a>	Problem starting the Workers.

Implements [BiometricEvaluation::Process::Manager](#).

**int32\_t BiometricEvaluation::Process::POSIXThreadManager::stopWorker ( std::shared\_ptr< WorkerController > workerController ) [virtual]**

Ask [Worker](#) to exit.

## Parameters

<i>worker↔ Controller</i>	Pointer to the <a href="#">WorkerController</a> that should be stopped.
-------------------------------	---

## Returns

Exit status of worker.

## Exceptions

<a href="#"><i>Error::ObjectDoesNot↔ Exist</i></a>	worker is not working.
<a href="#"><i>Error::StrategyError</i></a>	Problem sending the signal.

Implements [BiometricEvaluation::Process::Manager](#).

**void BiometricEvaluation::Process::POSIXThreadManager::waitForWorkerExit ( ) [virtual]**

Block until all Workers have exited.

Use this method if wait=false was set during a call to startWorker(s) but now wait=true is desired.

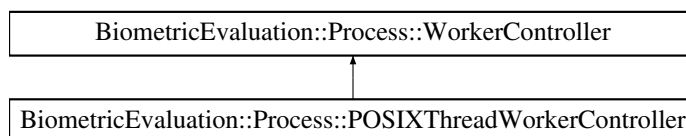
Implements [BiometricEvaluation::Process::Manager](#).

## F.85 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](#) ()  
*Reuse the [Worker](#).*
- bool [isWorking](#) () const  
*Obtain whether or not [Worker](#) is working.*
- bool [everWorked](#) () const  
*Obtain whether or not this [Worker](#) has ever worked.*
- [~POSIXThreadWorkerController](#) ()  
*[POSIXThreadWorkerController](#) destructor.*

## Friends

- class [POSIXThreadManager](#)

## Additional Inherited Members

### F.85.1 Detailed Description

Decorated [Worker](#) returned from a [Process::POSIXThreadManager](#).

### F.85.2 Member Function Documentation

**bool BiometricEvaluation::Process::POSIXThreadWorkerController::everWorked ( ) const**  
**[virtual]**

Obtain whether or not this [Worker](#) has ever worked.

Returns

true the [Worker](#) has ever or is currently working, false otherwise.

Note

[reset\(\)](#) will change the result of this method.

Implements [BiometricEvaluation::Process::WorkerController](#).

**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](#).

**void BiometricEvaluation::Process::POSIXThreadWorkerController::reset ( ) [virtual]**

Reuse the [Worker](#).

Exceptions

<a href="#"><i>Error::ObjectExists</i></a>	The previously started <a href="#">Worker</a> is still running.
--	---

Reimplemented from [BiometricEvaluation::Process::WorkerController](#).

## F.86 BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate 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](#) &[fingerView](#), [FingerImageCode](#) &[segment](#), [Image::CoordinateSet](#) &[coordinates](#))

Construct a [PrintPositionCoordinate](#).

### Public Attributes

- [FingerImageCode](#) [fingerView](#)
- [FingerImageCode](#) [segment](#)
- [Image::CoordinateSet](#) [coordinates](#)

#### F.86.1 Detailed Description

Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

#### F.86.2 Constructor & Destructor Documentation

**BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::PrintPositionCoordinate** ( [FingerImageCode](#) & *fingerView*, [FingerImageCode](#) & *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).

#### F.86.3 Member Data Documentation

**Image::CoordinateSet** **BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::coordinates**

Two coordinates forming bounding box

**FingerImageCode** **BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::fingerView**

Full finger view being bounded

**FingerImageCode BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPosition↵  
Coordinate::segment**

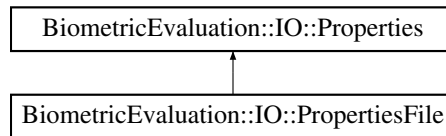
Segment within full finger view bound

## F.87 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 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)  
*Construct a new [Properties](#) object from the contents of a buffer.*
- virtual void [setProperty](#) (const std::string &property, const std::string &value)  
*Set a property with a value.*
- virtual void [setPropertyFromInteger](#) (const std::string &property, int64\_t value)  
*Set a property with an integer value.*
- virtual void [setPropertyFromDouble](#) (const std::string &property, double value)  
*Set a property with a double value.*
- virtual void [removeProperty](#) (const std::string &property)  
*Remove a property.*
- virtual std::string [getProperty](#) (const std::string &property) const  
*Retrieve a property value as a string object.*
- virtual int64\_t [getPropertyAsInteger](#) (const std::string &property) const  
*Retrieve a property value as an integer value.*
- virtual double [getPropertyAsDouble](#) (const std::string &property) const  
*Retrieve a property value as a double value.*
- std::vector< std::string > [getPropertyKeys](#) () const  
*Retrieve a set of all property keys.*
- virtual [~Properties](#) ()

### Protected Member Functions

- uint8\_t [getMode](#) () const  
*Obtain the mode of the [Properties](#) object.*
- void [initWithBuffer](#) (const [Memory::uint8Array](#) &buffer)  
*Initialize the PropertiesMap with the contents of a properly formatted buffer.*
- void [initWithBuffer](#) (const uint8\_t \*const buffer, size\_t size)  
*Initialize the PropertiesMap with the contents of a properly formatted buffer.*

## F.87.1 Detailed Description

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

## F.87.2 Constructor & Destructor Documentation

**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.
-----------	-------------	------------------------------------

**BiometricEvaluation::IO::Properties::Properties ( const uint8\_t \* *buffer*, const size\_t *size*, uint8\_t *mode* = *IO::READWRITE* )**

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.

Exceptions

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

**virtual BiometricEvaluation::IO::Properties::~~Properties ( ) [virtual]**

Destructor

## F.87.3 Member Function Documentation

**uint8\_t BiometricEvaluation::IO::Properties::getMode ( ) const [protected]**

Obtain the mode of the [Properties](#) object.

Returns

Mode (IO::READONLY or IO::READWRITE)

**virtual std::string BiometricEvaluation::IO::Properties::getProperty ( const std::string & *property* ) const [virtual]**

Retrieve a property value as a string object.

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.
----------------------------------	------------------------------------

**virtual double BiometricEvaluation::IO::Properties::getPropertyAsDouble ( const std::string & property ) const [virtual]**

Retrieve a property value as a double value.

Parameters

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

Exceptions

<i>Error::ObjectDoesNotExist</i>	The named property does not exist.
----------------------------------	------------------------------------

**virtual int64\_t BiometricEvaluation::IO::Properties::getPropertyAsInteger ( const std::string & property ) const [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	property	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.

**std::vector<std::string> BiometricEvaluation::IO::Properties::getPropertyKeys ( ) const**

Retrieve a set of all property keys.

Returns

A vector of property key strings.

**void BiometricEvaluation::IO::Properties::initWithBuffer ( const Memory::uint8Array & buffer ) [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

<a href="#"><i>Error::StrategyError</i></a>	A line of the buffer is malformed.
---	------------------------------------

**void BiometricEvaluation::IO::Properties::initWithBuffer ( const uint8\_t \*const *buffer*, size\_t *size* )**  
**[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

<a href="#"><i>Error::StrategyError</i></a>	A line of the buffer is malformed.
---	------------------------------------

**virtual void BiometricEvaluation::IO::Properties::removeProperty ( const std::string & *property* )**  
**[virtual]**

Remove a property.

Parameters

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

Exceptions

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

**virtual void BiometricEvaluation::IO::Properties::setProperty ( const std::string & *property*, const std::string & *value* )**  
**[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

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

Exceptions

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

**virtual void BiometricEvaluation::IO::Properties::setPropertyFromDouble ( const std::string & *property*, double *value* )**  
**[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

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

**virtual void BiometricEvaluation::IO::Properties::setPropertyFromInteger ( const std::string &property, int64\_t value ) [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

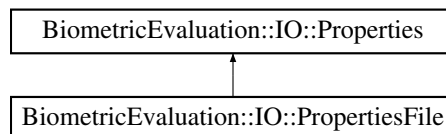
<a href="#"><i>Error::StrategyError</i></a>	The <a href="#">Properties</a> object is read-only.
---	---

## F.88 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 std::string &filename, uint8\_t mode=IO::READWRITE)  
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](#) ()  
Write the properties to the underlying file, synchronizing the in-memory and on-disk versions.
- void [changeName](#) (const std::string &filename)  
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](#) ()
- [PropertiesFile](#) (const [PropertiesFile](#) &other)=delete  
Copy constructor (disabled).
- [PropertiesFile](#) & operator= (const [PropertiesFile](#) &other)=delete  
Assignment operator (disabled).

## Additional Inherited Members

### F.88.1 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.

### F.88.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::PropertiesFile::PropertiesFile ( const std::string &filename, uint8\_t mode = IO::READWRITE )**

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.

Exceptions

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

**BiometricEvaluation::IO::PropertiesFile::~~PropertiesFile ( )**

Destructor

**BiometricEvaluation::IO::PropertiesFile::PropertiesFile ( const PropertiesFile &other ) [delete]**

Copy constructor (disabled).

Disabled because this object could represent a file on disk.

Parameters

<i>other</i>	<a href="#">PropertiesFile</a> object to copy.
--------------	--

### F.88.3 Member Function Documentation

**void BiometricEvaluation::IO::PropertiesFile::changeName ( const std::string &filename )**

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

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

## Exceptions

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

### PropertiesFile& BiometricEvaluation::IO::PropertiesFile::operator= ( const PropertiesFile & *other* ) [`delete`]

Assignment operator (disabled).

Disabled because this object could represent a file on disk.

## Parameters

<code>other</code>	<a href="#">PropertiesFile</a> object to assign;
--------------------	--

## Returns

This [PropertiesFile](#) object, now containing the contents of *other*.

### void BiometricEvaluation::IO::PropertiesFile::sync ( )

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

## Exceptions

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

## F.89 BiometricEvaluation::Iris::INCITSView::QualitySubBlock Struct Reference

Representation of an iris quality block.

```
#include <be_iris_incitsview.h>
```

### Public Attributes

- `uint8_t score`
- `uint16_t vendorID`
- `uint16_t algorithmID`

### F.89.1 Detailed Description

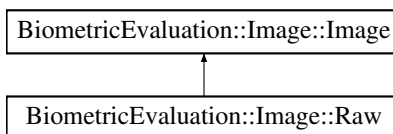
Representation of an iris quality block.

## F.90 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::uint8Array](#) [getRawData](#) () const  
*Accessor for the raw image data. The data returned should not be compressed or encoded.*
- [Memory::uint8Array](#) [getRawGrayscaleData](#) (uint8\_t depth=8) const  
*Accessor for decompressed data in grayscale.*

### Additional Inherited Members

#### F.90.1 Detailed Description

An image with no encoding or compression.

#### F.90.2 Member Function Documentation

**[Memory::uint8Array](#) [BiometricEvaluation::Image::Raw::getRawData](#) ( ) const [\[virtual\]](#)**

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

Returns

AutoArray holding raw image data.

Exceptions

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

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

**[Memory::uint8Array](#) [BiometricEvaluation::Image::Raw::getRawGrayscaleData](#) ( uint8\_t depth = 8 ) const [\[virtual\]](#)**

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

AutoArray holding raw grayscale image data.

## 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](#).

## F.91 BiometricEvaluation::MPI::Receiver Class Reference

A class to represent an [MPI](#) task that receives WorkPackages containers from the [Distributor](#).

```
#include <be_mpi_receiver.h>
```

### Public Member Functions

- [Receiver](#) (const std::string &propertiesFileName, const std::shared\_ptr< [BiometricEvaluation::MPI::WorkPackageProcessor](#) > &workPackageProcessor)  
*Construct a new work package receiver.*
- void [start](#) ()  
*Start the receiving task.*

### F.91.1 Detailed Description

A class to represent an [MPI](#) task that receives WorkPackages containers from the [Distributor](#).

A receiver object depends on a set of properties contained in a file. The properties specify [MPI](#) settings, and other items. Subclasses of the class can add new properties.

Each receiver object is responsible for 1..n worker processes that are started when [Receiver::start\(\)](#) is called. The receiver will start workers only when the distributor indicates that it has started successfully. Otherwise, the [Receiver](#) transitions to the shutdown state.

One of the optional properties is a Uniform Resource Locator (URL) for the Logsheet. If this property does not exist, no logging takes place (although applications can create their own Logsheet). If the URL is present, the framework will log at various points of processing. In the case of a FileLogsheet the framework will create more than one log file, each named after the ID of the [MPI](#) task created by the [MPI](#) runtime, and the child process created by [Receiver](#).

See also

[IO::Properties](#)  
[IO::Logsheet](#)  
[MPI::Distributor](#)  
[Process::Worker](#)

### F.91.2 Constructor & Destructor Documentation

**BiometricEvaluation::MPI::Receiver::Receiver** ( *const* std::string & *propertiesFileName*, *const* std::shared\_ptr< BiometricEvaluation::MPI::WorkPackageProcessor > & *workPackageProcessor* )

Construct a new work package receiver.

## Parameters

in	<i>propertiesFile↔ Name</i>	The name of the file containing the properties used by the receiver object.
in	<i>workPackage↔ Processor</i>	The object that will process the work received by this object.

## Exceptions

<i>Error::Exception</i>	An error occurred when constructing this object.
-------------------------	--

## F.91.3 Member Function Documentation

**void BiometricEvaluation::MPI::Receiver::start ( )**

Start the receiving task.

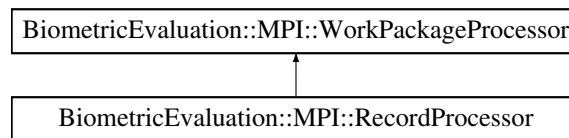
Upon starting, the [Receiver](#) object will begin communicating with the [Distributor](#) using [MPI](#) messages. This [Receiver](#) object will send a status message back to the [Distributor](#) indicating success or failure to initialize. Success includes the startup of at least one worker process.

## F.92 BiometricEvaluation::MPI::RecordProcessor Class Reference

An implementation of a work package processor that will extract record store keys, and optionally, values, from a [WorkPackage](#).

```
#include <be_mpi_recordprocessor.h>
```

Inheritance diagram for BiometricEvaluation::MPI::RecordProcessor:



## Public Member Functions

- [RecordProcessor](#) (const std::string &propertiesFileName)  
*Construct a work package processor with the given properties.*
- virtual void [processRecord](#) (const std::string &key)=0  
*Method implemented by child classes to perform an action using each record from the Record Store.*
- virtual void [processRecord](#) (const std::string &key, const [Memory::uint8Array](#) &value)=0  
*Method implemented by child classes to perform an action using each record from the Record Store.*
- virtual std::shared\_ptr  
< [WorkPackageProcessor](#) > [newProcessor](#) (std::shared\_ptr< [IO::Logsheet](#) > &logsheet)=0  
*Obtain an object that will process work packages. This method is part of the factory personality.*
- virtual void [performInitialization](#) (std::shared\_ptr< [IO::Logsheet](#) > &logsheet)=0  
*Initialization function to be called before work is distributed to the work package processor.*
- void [processWorkPackage](#) ([MPI::WorkPackage](#) &workPackage)  
*Process the data contents of the work package. This method is part of the worker personality.*



## Protected Member Functions

- `std::shared_ptr`  
< [MPI::RecordStoreResources](#) > `getResources ()`

### F.92.1 Detailed Description

An implementation of a work package processor that will extract record store keys, and optionally, values, from a [WorkPackage](#).

Subclasses of this abstract class must implement the method to process the records associated with the keys.

### F.92.2 Constructor & Destructor Documentation

**BiometricEvaluation::MPI::RecordProcessor::RecordProcessor ( const std::string & *propertiesFileName* )**

Construct a work package processor with the given properties.

A record processor uses a named record store to retrieve the data to be processed when only the key is delivered as part of a work package. When both key and value are part of the work package, there is no need to have access to the source record store.

Note

The size of a single value item is limited to  $2^{32}$  octets. If the size of the value item is larger, behavior is undefined.

Parameters

<code>in</code>	<i>propertiesFile↔ Name</i>	The name of the file containing the properties for this object.
-----------------	---------------------------------	---

Exceptions

<a href="#">Error::Exception</a>	An error occurred, usually due to missing or incorrect properties.
----------------------------------	--

### F.92.3 Member Function Documentation

**virtual std::shared\_ptr<WorkPackageProcessor> BiometricEvaluation::MPI::Record↔  
Processor::newProcessor ( std::shared\_ptr< IO::Logsheet > & *logsheet* ) [pure  
virtual]**

Obtain an object that will process work packages. This method is part of the factory personality.

Parameters

<i>logsheet</i>	A shared pointer to the <a href="#">IO::Logsheet</a> that may be used to save messages generated by the object.
-----------------	---

Returns

A shared pointer to the work package processor.

Implements [BiometricEvaluation::MPI::WorkPackageProcessor](#).

**virtual void BiometricEvaluation::MPI::RecordProcessor::performInitialization ( std::shared\_ptr< IO::Logsheet > &logsheet ) [pure virtual]**

Initialization function to be called before work is distributed to the work package processor.

Implementations of this class can use this function to do any processing necessary before work is given to the processor, pre-forking.

This method is part of the factory personality. All state that is to be common across all package processor objects can be initialized in this method.

Parameters

<i>logsheet</i>	A shared pointer to the <a href="#">IO::Logsheet</a> that may be used to save messages generated by the object.
-----------------	---

Exceptions

<a href="#">Error::Exception</a>	An implementation specific. error occurred.
----------------------------------	---

Implements [BiometricEvaluation::MPI::WorkPackageProcessor](#).

**virtual void BiometricEvaluation::MPI::RecordProcessor::processRecord ( const std::string &key ) [pure virtual]**

Method implemented by child classes to perform an action using each record from the Record Store.

The source RecordStore must be accessible to the the implementation as the value for each key is not included.

Parameters

in	<i>key</i>	The key associated with the record that is to be processed.
----	------------	---

Exceptions

<a href="#">Error::Exception</a>	An error occurred processing the record: Missing record, input/output error, or memory allocation.
----------------------------------	--

**virtual void BiometricEvaluation::MPI::RecordProcessor::processRecord ( const std::string &key, const Memory::uint8Array &value ) [pure virtual]**

Method implemented by child classes to perform an action using each record from the Record Store.

Parameters

in	<i>key</i>	The key associated with the record that is to be processed.
in	<i>value</i>	The data from the record that is to be processed.

Exceptions

<a href="#">Error::Exception</a>	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.
----------------------------------	--

**void BiometricEvaluation::MPI::RecordProcessor::processWorkPackage ( MPI::WorkPackage &workPackage ) [virtual]**

[Process](#) the data contents of the work package. This method is part of the worker personality.

## Parameters

<code>in</code>	<code>workPackage</code>	The work package.
-----------------	--------------------------	-------------------

## Exceptions

<a href="#"><i>Error::Exception</i></a>	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.
---	--

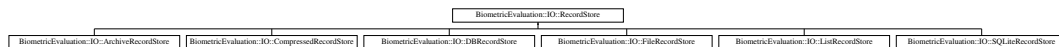
Implements [BiometricEvaluation::MPI::WorkPackageProcessor](#).

## F.93 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 Types

- enum [Kind](#) {  
[Kind::BerkeleyDB](#), [Kind::Archive](#), [Kind::File](#), [Kind::SQLite](#),  
[Kind::Compressed](#), [Kind::List](#), [Kind::Default](#) = [BerkeleyDB](#) }
- using [iterator](#) = [IO::RecordStoreIterator](#)
- using [const\\_iterator](#) = const [IO::RecordStoreIterator](#)

### Public Member Functions

- `std::string` [getDescription](#) () const
- unsigned int [getCount](#) () const
- `std::string` [getPathname](#) () const
- virtual void [move](#) (const `std::string` &pathname)
- *Move the [RecordStore](#).*
- virtual void [changeDescription](#) (const `std::string` &description)
- virtual `uint64_t` [getSpaceUsed](#) () const
- *Obtain real storage utilization.*
- virtual void [sync](#) () const
- virtual void [insert](#) (const `std::string` &key, const void \*const data, const `uint64_t` size)=0
- virtual void [insert](#) (const `std::string` &key, const [Memory::uint8Array](#) &data)
- virtual void [remove](#) (const `std::string` &key)=0
- virtual `uint64_t` [read](#) (const `std::string` &key, void \*const data) const =0
- virtual `uint64_t` [read](#) (const `std::string` &key, [Memory::uint8Array](#) &data) const
- *Read a complete record from a store.*
- virtual void [replace](#) (const `std::string` &key, const void \*const data, const `uint64_t` size)=0
- virtual void [replace](#) (const `std::string` &key, const [Memory::uint8Array](#) &data)
- virtual `uint64_t` [length](#) (const `std::string` &key) const =0
- virtual void [flush](#) (const `std::string` &key) const =0
- virtual `uint64_t` [sequence](#) (`std::string` &key, void \*const data=nullptr, int cursor=[BE\\_RECSTORE\\_SEQUENCE\\_NEXT](#))=0

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

- virtual uint64\_t [sequence](#) (std::string &key, [Memory::uint8Array](#) &data, int cursor=[BE\\_RECSTORE\\_SEQ\\_NEXT](#))

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

- virtual void [setCursorAtKey](#) (const std::string &key)=0
- virtual bool [containsKey](#) (const std::string &key) const

*Determines whether the [RecordStore](#) contains an element with the specified key.*

- virtual [iterator begin](#) () noexcept
- virtual [iterator end](#) () noexcept

## Static Public Member Functions

- static std::shared\_ptr  
< [RecordStore](#) > [openRecordStore](#) (const std::string &pathname, uint8\_t mode=READWRITE)  
*Open an existing [RecordStore](#) and return a managed pointer to the the object representing that store.*
- static std::shared\_ptr  
< [RecordStore](#) > [createRecordStore](#) (const std::string &pathname, const std::string &description, const [Kind](#) &kind)  
*Create a new [RecordStore](#) and return a managed pointer to the the object representing that store.*
- static void [removeRecordStore](#) (const std::string &pathname)
- static void [mergeRecordStores](#) (const std::string &mergePathname, const std::string &description, const [RecordStore::Kind](#) &kind, const std::vector< std::string > &pathnames)  
*Create a new [RecordStore](#) that contains the contents of several other [RecordStores](#).*

## Static Public Attributes

- static const std::string [INVALIDKEYCHARS](#)
- static const char [KEY\\_SEGMENT\\_SEPARATOR](#) = '&'
- static const uint64\_t [KEY\\_SEGMENT\\_START](#) = 1
- static const std::string [CONTROLFILENAME](#)
- static const std::string [DESCRIPTIONPROPERTY](#)
- static const std::string [COUNTPROPERTY](#)
- static const std::string [TYPEPROPERTY](#)
- static const std::string [RSREADONLYERROR](#)
- static const int [BE\\_RECSTORE\\_SEQ\\_START](#) = 1
- static const int [BE\\_RECSTORE\\_SEQ\\_NEXT](#) = 2

## Protected Member Functions

- [RecordStore](#) (const std::string &pathname, const std::string &description, const [Kind](#) &kind)
- [RecordStore](#) (const std::string &pathname, uint8\_t mode=READWRITE)
- uint8\_t [getMode](#) () const
- std::string [canonicalName](#) (const std::string &name) const
- int [getCursor](#) () const
- void [setCursor](#) (int cursor)
- bool [validateKeyString](#) (const std::string &key) const
- void [setProperty](#) (const std::shared\_ptr< [IO::Properties](#) > properties)  
*Replace existing [Properties](#) in [RecordStore](#) control file.*
- std::shared\_ptr< [IO::Properties](#) > [getProperties](#) () const  
*Obtain a copy of the [Properties](#) object.*

## Static Protected Member Functions

- static std::string [genKeySegName](#) (const std::string &key, const uint64\_t segnum)  
*Generate key segment names.*

### F.93.1 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](#).

### F.93.2 Member Enumeration Documentation

**enum BiometricEvaluation::IO::RecordStore::Kind** [**strong**]

Possible types of [RecordStore](#)

Enumerator

*BerkeleyDB* [DBRecordStore](#)  
*Archive* [ArchiveRecordStore](#)  
*File* [FileRecordStore](#)  
*SQLite* [SQLiteRecordStore](#)  
*Compressed* [CompressedRecordStore](#)  
*List* [ListRecordStore](#)  
*Default* "Default" [RecordStore](#) kind

### F.93.3 Constructor & Destructor Documentation

**BiometricEvaluation::IO::RecordStore::RecordStore** ( const std::string &*pathname*, const std::string &*description*, const Kind &*kind* ) [**protected**]

Constructor to create a new [RecordStore](#).

Parameters

in	<i>pathname</i>	The pathname where the <a href="#">RecordStore</a> is to be created.
in	<i>description</i>	The text used to describe the store.
in	<i>kind</i>	The kind of <a href="#">RecordStore</a> .

Exceptions

<a href="#">Error::ObjectExists</a>	The store was previously created, or the directory where it would be created exists.
-------------------------------------	--

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

**BiometricEvaluation::IO::RecordStore::RecordStore** ( *const std::string & pathname*, *uint8\_t mode = READWRITE* ) [**protected**]

Constructor to open an existing [RecordStore](#).

Parameters

<i>in</i>	<i>pathname</i>	The pathname where the <a href="#">RecordStore</a> is to be created.
<i>in</i>	<i>mode</i>	The type of access a client of this <a href="#">RecordStore</a> has.

Exceptions

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

### F.93.4 Member Function Documentation

**virtual iterator BiometricEvaluation::IO::RecordStore::begin** ( ) [**virtual**], [**noexcept**]

Returns

Iterator to the first record.

**virtual void BiometricEvaluation::IO::RecordStore::changeDescription** ( *const std::string & description* ) [**virtual**]

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

Parameters

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

Exceptions

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

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

**virtual bool BiometricEvaluation::IO::RecordStore::containsKey** ( *const std::string & key* ) **const** [**virtual**]

Determines whether the [RecordStore](#) contains an element with the specified key.

Parameters

<i>key</i>	The key to locate.
------------	--------------------

Returns

True if the [RecordStore](#) contains an element with the key, false otherwise.

```
static std::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::createRecordStore (  
const std::string & pathname, const std::string & description, const Kind & kind ) [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>pathname</i>	The directory of the store to be created.
in	<i>description</i>	The description of the store to be created.
in	<i>kind</i>	The kind of <a href="#">RecordStore</a> to be created.

## Returns

An managed pointer to the object representing the created store.

## Exceptions

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

**virtual iterator BiometricEvaluation::IO::RecordStore::end ( ) [virtual], [noexcept]**

## Returns

Iterator past the last record.

**virtual void BiometricEvaluation::IO::RecordStore::flush ( const std::string & key ) const [pure virtual]**

Commit the record's data to storage.

## Parameters

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

## Exceptions

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

Implemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**static std::string BiometricEvaluation::IO::RecordStore::genKeySegName ( const std::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.



**unsigned int BiometricEvaluation::IO::RecordStore::getCount ( ) const**

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

Returns

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

**std::string BiometricEvaluation::IO::RecordStore::getDescription ( ) const**

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

Returns

The [RecordStore](#)'s description.

**std::string BiometricEvaluation::IO::RecordStore::getPathname ( ) const**

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

Returns

Where in the file system the [RecordStore](#) is located.

**std::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.

**virtual uint64\_t BiometricEvaluation::IO::RecordStore::getSpaceUsed ( ) const [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><a href="#">Error::StrategyError</a></i>	An error occurred when using the underlying storage system.
---	---

Reimplemented in [BiometricEvaluation::IO::ListRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**virtual void BiometricEvaluation::IO::RecordStore::insert ( const std::string & key, const void \*const data, const uint64\_t size ) [pure virtual]**

Insert a record into the store.

## Parameters

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

## Exceptions

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

Implemented in [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**virtual void BiometricEvaluation::IO::RecordStore::insert ( const std::string & *key*, const Memory::uint8Array & *data* ) [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

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

**virtual uint64\_t BiometricEvaluation::IO::RecordStore::length ( const std::string & *key* ) const [pure virtual]**

Return the length of a record.

## Parameters

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

## Returns

The record length.

## Exceptions

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

Implemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**static void BiometricEvaluation::IO::RecordStore::mergeRecordStores ( const std::string & *mergePathname*, const std::string & *description*, const RecordStore::Kind & *kind*, const std::vector< std::string > & *pathnames* ) [static]**

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

## Parameters

in	<i>mergePathname</i>	The path name of the new <a href="#">RecordStore</a> that will be created.
in	<i>description</i>	The text used to describe the new <a href="#">RecordStore</a> .
in	<i>kind</i>	The kind of the new, merged <a href="#">RecordStore</a> .
in	<i>pathnames</i>	Vector of path names to RecordStores to open. These are the RecordStores that will be merged to create the new <a href="#">RecordStore</a> .

## Exceptions

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

**virtual void BiometricEvaluation::IO::RecordStore::move ( const std::string & *pathname* )**  
**[virtual]**

Move the [RecordStore](#).

The [RecordStore](#) can be moved to a new path in the file system.

## Parameters

in	<i>pathname</i>	The new path of the <a href="#">RecordStore</a> .
----	-----------------	---

## Exceptions

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

Reimplemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**static std::shared\_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::openRecordStore ( const std::string & *pathname*, uint8\_t *mode* = *READWRITE* )**  
**[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>pathname</i>	The path name of the store to be opened.
in	<i>mode</i>	The type of access a client of this <a href="#">RecordStore</a> has.

## Returns

An object representing the existing store.

## Exceptions

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

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

**virtual uint64\_t BiometricEvaluation::IO::RecordStore::read ( const std::string & key, void \*const data ) const [pure virtual]**

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

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

Returns

The size of the record.

Exceptions

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

Implemented in [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**virtual uint64\_t BiometricEvaluation::IO::RecordStore::read ( const std::string & key, Memory::uint8Array & data ) const [virtual]**

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

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

Returns

The size of the record.

Exceptions

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

**virtual void BiometricEvaluation::IO::RecordStore::remove ( const std::string & key ) [pure virtual]**

Remove a record from the store.

## Parameters

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

## Exceptions

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

Implemented in [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**static void BiometricEvaluation::IO::RecordStore::removeRecordStore ( const std::string & pathname ) [static]**

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

## Parameters

in	pathname	The name of the existing <a href="#">RecordStore</a> .
----	----------	--

## Exceptions

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

**virtual void BiometricEvaluation::IO::RecordStore::replace ( const std::string & key, const void \*const data, const uint64\_t size ) [pure virtual]**

Replace a complete record in a store.

## Parameters

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

## Exceptions

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

Implemented in [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**virtual void BiometricEvaluation::IO::RecordStore::replace ( const std::string & key, const Memory::uint8Array & data ) [virtual]**

Replace a complete record in a [RecordStore](#).

## Parameters

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

## Exceptions

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

**virtual uint64\_t BiometricEvaluation::IO::RecordStore::sequence ( std::string & key, void \*const data = nullptr, int cursor = BE\_RECSTORE\_SEQ\_NEXT ) [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	key	The key of the currently sequenced record.
in	data	Pointer to where the data is to be written. Applications can set data to nullptr to indicate only the key is wanted.
in	cursor	The location within the sequence of the key/data pair to return.

## Returns

The length of the record currently in sequence.

## Exceptions

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

Implemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**virtual uint64\_t BiometricEvaluation::IO::RecordStore::sequence ( std::string & key, Memory::uint8Array & data, int cursor = BE\_RECSTORE\_SEQ\_NEXT ) [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	<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

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

**virtual void BiometricEvaluation::IO::RecordStore::setCursorAtKey ( const std::string & key )**  
**[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 <a href="#">sequence()</a> .
----	------------	---

## Exceptions

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

Implemented in [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::DBRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), [BiometricEvaluation::IO::FileRecordStore](#), and [BiometricEvaluation::IO::SQLiteRecordStore](#).

**void BiometricEvaluation::IO::RecordStore::setProperties ( const std::shared\_ptr< IO::Properties > properties )**  
**[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 <a href="#">Properties</a> object.
----	-------------------	--

## Exceptions

<a href="#"><i>Error::StrategyError</i></a>	<a href="#">RecordStore</a> was opened READONLY.
---	--

**virtual void BiometricEvaluation::IO::RecordStore::sync ( ) const** **[virtual]**

Synchronize the entire record store to persistent storage.

## Exceptions

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

Reimplemented in [BiometricEvaluation::IO::ArchiveRecordStore](#), [BiometricEvaluation::IO::CompressedRecordStore](#), [BiometricEvaluation::IO::ListRecordStore](#), and [BiometricEvaluation::IO::DBRecordStore](#).

### F.93.5 Member Data Documentation

**const int BiometricEvaluation::IO::RecordStore::BE\_RECSTORE\_SEQ\_NEXT = 2 [static]**

Tell sequence to sequence from current position

**const int BiometricEvaluation::IO::RecordStore::BE\_RECSTORE\_SEQ\_START = 1 [static]**

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

**const std::string BiometricEvaluation::IO::RecordStore::CONTROLFILENAME [static]**

The name of the control file, a properties list

**const std::string BiometricEvaluation::IO::RecordStore::COUNTPROPERTY [static]**

Property key for the number of store items

**const std::string BiometricEvaluation::IO::RecordStore::DESCRIPTIONPROPERTY [static]**

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

**const std::string BiometricEvaluation::IO::RecordStore::INVALIDKEYCHARS [static]**

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

**const char BiometricEvaluation::IO::RecordStore::KEY\_SEGMENT\_SEPARATOR = '&' [static]**

Character used to separate key segments

**const uint64\_t BiometricEvaluation::IO::RecordStore::KEY\_SEGMENT\_START = 1 [static]**

First segment number of a segmented record

**const std::string BiometricEvaluation::IO::RecordStore::RSREADONLYERROR [static]**

Message for READONLY [RecordStore](#) modification

**const std::string BiometricEvaluation::IO::RecordStore::TYPEPROPERTY [static]**

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

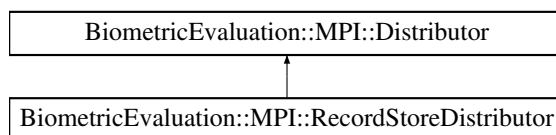


## F.94 BiometricEvaluation::MPI::RecordStoreDistributor Class Reference

An implementation of the Distributor abstraction that uses a record store for input to create the work packages.

```
#include <be_mpi_recordstoredistributor.h>
```

Inheritance diagram for BiometricEvaluation::MPI::RecordStoreDistributor:



### Public Member Functions

- [RecordStoreDistributor](#) (const std::string &propertiesFileName, const bool includeValues)

*Construct a distributor using the named properties.*

### Protected Member Functions

- void [createWorkPackage](#) (MPI::WorkPackage &workPackage)

*Create a work package for distribution.*

### F.94.1 Detailed Description

An implementation of the Distributor abstraction that uses a record store for input to create the work packages.

### F.94.2 Constructor & Destructor Documentation

**BiometricEvaluation::MPI::RecordStoreDistributor::RecordStoreDistributor ( const std::string &propertiesFileName, const bool includeValues )**

Construct a distributor using the named properties.

The distributor object is based on the properties given in the file. The name of the input record store must be one of the properties.

The work package sent to Receivers can contain either RecordStore keys, or key/value pairs.

Note

The size of a single value item is limited to  $2^{32}$  octets. If the size of the value item is larger, behavior is undefined.

Parameters

in	<i>propertiesFileName</i>	The file containing the properties.
----	---------------------------	-------------------------------------

<code>in</code>	<code>includeValues</code>	true if both the key and value items are included in the work package, false otherwise.
-----------------	----------------------------	---

Exceptions

<a href="#"><i>Error::Exception</i></a>	An error occurred, typically due to missing or invalid properties.
---	--

See also

[MPI::Distributor](#)  
[MPI::RecordProcessor](#)  
[MPI::RecordStoreResources](#)

### F.94.3 Member Function Documentation

**void BiometricEvaluation::MPI::RecordStoreDistributor::createWorkPackage ( MPI::WorkPackage & workPackage ) [protected], [virtual]**

Create a work package for distribution.

Implementations of this class create a work package to encapsulate the specific data type that is to be distributed.

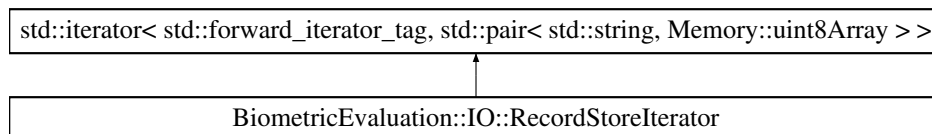
Implements [BiometricEvaluation::MPI::Distributor](#).

## F.95 BiometricEvaluation::IO::RecordStoreIterator Class Reference

Generic ForwardIterator for all RecordStores.

```
#include <be_io_recordstoreiterator.h>
```

Inheritance diagram for BiometricEvaluation::IO::RecordStoreIterator:



### Public Member Functions

- [RecordStoreIterator](#) ()  
*Default constructor.*
- [RecordStoreIterator](#) (IO::RecordStore \*recordStore, bool atEnd=false)  
*Constructor.*
- [RecordStoreIterator](#) (const [RecordStoreIterator](#) &rhs)=default
- [RecordStoreIterator](#) ([RecordStoreIterator](#) &&rvalue)=default
- virtual [~RecordStoreIterator](#) ()=default
- reference [operator\\*](#) ()
- pointer [operator->](#) ()
- [RecordStoreIterator](#) [operator++](#) ()
- [RecordStoreIterator](#) [operator++](#) (int postfix)
- [RecordStoreIterator](#) [operator+=](#) (difference\_type rhs)

*Advance a variable number of arguments.*

- [RecordStoreIterator](#) **operator+** (difference\_type rhs)  
*Advance a variable number of arguments.*
- bool **operator==** (const [RecordStoreIterator](#) &rhs)  
*Equivalence operator.*
- bool **operator!=** (const [RecordStoreIterator](#) &rhs)  
*Non-equivalence operator.*
- [RecordStoreIterator](#) & **operator=** ([RecordStoreIterator](#) &rhs)=default
- [RecordStoreIterator](#) & **operator=** ([RecordStoreIterator](#) &&rhs)=default

### F.95.1 Detailed Description

Generic ForwardIterator for all RecordStores.

Note

Dereferencing an iterator returns a copy of the value. Modifying a non-const iterator does not manipulate the underlying [RecordStore](#).

This generic iterator provides no optimization over [RecordStore::sequence\(\)](#).

### F.95.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator ( )**

Default constructor.

Creates "end" iterator.

Note

Satisfies DefaultConstructible requirement.

**BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator ( IO::RecordStore \* recordStore, bool atEnd = false )**

Constructor.

Parameters

<i>recordStore</i>	Pointer to a <a href="#">RecordStore</a> that will be iterated over.
<i>atEnd</i>	Whether or not to start at the "end" iterator.

Note

Iterator defaults to starting at the beginning of the [RecordStore](#).

[RecordStoreIterator](#) does not retain any ownership of recordStore.

**BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator ( const RecordStoreIterator & rhs ) [default]**

Default copy constructor

**BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator ( RecordStoreIterator && rvalue ) [default]**

Default move constructor

**virtual BiometricEvaluation::IO::RecordStoreIterator::~~RecordStoreIterator ( )** [**virtual**],  
[**default**]

Default destructor

### F.95.3 Member Function Documentation

**bool BiometricEvaluation::IO::RecordStoreIterator::operator!= ( const RecordStoreIterator & rhs )**  
[**inline**]

Non-equivalence operator.

Parameters

<i>rhs</i>	Reference to <a href="#">RecordStoreIterator</a> being compared.
------------	--

Returns

Whether or not this is not equivalent to rhs.

Note

Satisfies "i != j" is equivalent to "!(i == j)" condition of InputIterator.

**reference BiometricEvaluation::IO::RecordStoreIterator::operator\* ( )**

Returns

Reference to a key/value pair.

**RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator+ ( difference\_type rhs )**

Advance a variable number of arguments.

Parameters

<i>rhs</i>	Number of objects to advance (1 or more).
------------	---

Returns

Self after advancing rhs objects.

**RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator++ ( )**

Returns

Self after advancing.

**RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator++ ( int postfix )**

Returns

Copy of self before advancing.

**RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator+= ( difference\_type rhs )**

Advance a variable number of arguments.

Parameters

<i>rhs</i>	Number of objects to advance (1 or more).
------------	---

Returns

Self after advancing rhs objects.

**pointer** `BiometricEvaluation::IO::RecordStoreIterator::operator-> ( )`

Returns

A dereferenced key/value pair.

**RecordStoreIterator&** `BiometricEvaluation::IO::RecordStoreIterator::operator= ( RecordStoreIterator && rhs ) [default]`

Default move assignment operator

**bool** `BiometricEvaluation::IO::RecordStoreIterator::operator== ( const RecordStoreIterator & rhs )`

Equivalence operator.

Parameters

<i>rhs</i>	Reference to <a href="#">RecordStoreIterator</a> being compared.
------------	--

Returns

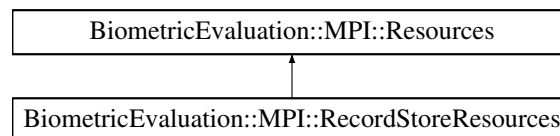
Whether or not this is equivalent to rhs.

## F.96 BiometricEvaluation::MPI::RecordStoreResources Class Reference

A class to represent a set of resources needed by an [MPI](#) program using a RecordStore for input.

```
#include <be_mpi_recordstoreresources.h>
```

Inheritance diagram for BiometricEvaluation::MPI::RecordStoreResources:



### Public Member Functions

- [RecordStoreResources](#) (const std::string &propertiesFileName)  
*Constructor taking the name of the properties file with the resource names.*
- uint32\_t [getChunkSize](#) () const
- bool [haveRecordStore](#) () const  
*Indicator that a record store has been opened.*
- std::shared\_ptr< [IO::RecordStore](#) > [getRecordStore](#) () const  
*Return the RecordStore named in the property set.*

## Static Public Member Functions

- static std::vector< std::string > [getRequiredProperties](#) ()  
*Obtain the required properties as strings.*
- static std::vector< std::string > [getOptionalProperties](#) ()  
*Obtain the list of optional properties.*

## Static Public Attributes

- static const std::string [INPUTRSPROPERTY](#)  
*The property string “Input Record Store”; required.*
- static const std::string [CHUNKSIZEPROPERTY](#)  
*The property string “Chunk Size”; required.*

### F.96.1 Detailed Description

A class to represent a set of resources needed by an [MPI](#) program using a RecordStore for input.

[Resources](#) are opened based on the property when appropriate. The input record store need not be accessible. Applications should call [haveRecordStore\(\)](#) to check whether the record store has been opened.

### F.96.2 Constructor & Destructor Documentation

**BiometricEvaluation::MPI::RecordStoreResources::RecordStoreResources ( const std::string & propertiesFileName )**

Constructor taking the name of the properties file with the resource names.

Exceptions

<a href="#">Error::FileError</a>	The resources file could not be read.
<a href="#">Error::ObjectDoesNotExist</a>	A required property does not exist.
<a href="#">Error::Exception</a>	Some other error occurred.

### F.96.3 Member Function Documentation

**static std::vector<std::string> BiometricEvaluation::MPI::RecordStoreResources::getOptionalProperties ( ) [static]**

Obtain the list of optional properties.

Returns

A set of optional property strings.

**std::shared\_ptr<IO::RecordStore> BiometricEvaluation::MPI::RecordStoreResources::getRecordStore ( ) const**

Return the RecordStore named in the property set.

Returns

A shared pointer to the record store.

```
static std::vector<std::string> BiometricEvaluation::MPI::RecordStoreResources::getRequiredProperties ( ) [static]
```

Obtain the required properties as strings.

Returns

The set of required properties.

```
bool BiometricEvaluation::MPI::RecordStoreResources::haveRecordStore ( ) const
```

Indicator that a record store has been opened.

Returns

true if input record store is opened, false otherwise.

## F.97 BiometricEvaluation::Image::Resolution Struct Reference

A structure to represent the resolution of an image.

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

### Public Types

- enum [Units](#) { [Units::NA](#) = 0, [Units::PPI](#) = 1, [Units::PPMM](#) = 2, [Units::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 [Units](#) [units](#)=[Units::PPI](#))

Create a [Resolution](#) struct.

### Public Attributes

- double [xRes](#)
- double [yRes](#)
- [Units](#) [units](#)

#### F.97.1 Detailed Description

A structure to represent the resolution of an image.

#### F.97.2 Member Enumeration Documentation

```
enum BiometricEvaluation::Image::Resolution::Units [strong]
```

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

### F.97.3 Constructor & Destructor Documentation

**BiometricEvaluation::Image::Resolution::Resolution** ( `const double xRes = 0.0`, `const double yRes = 0.0`, `const Units units = Units::PPI` )

Create a [Resolution](#) struct.



Parameters

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

## F.97.4 Member Data Documentation

### Units BiometricEvaluation::Image::Resolution::units

Units in which xRes and yRes are represented

### double BiometricEvaluation::Image::Resolution::xRes

[Resolution](#) along the X-axis

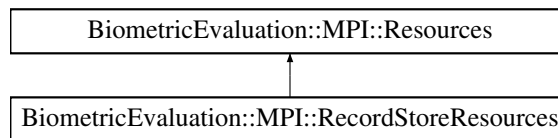
### double BiometricEvaluation::Image::Resolution::yRes

[Resolution](#) along the Y-axis

## F.98 BiometricEvaluation::MPI::Resources Class Reference

```
#include <be_mpi_resources.h>
```

Inheritance diagram for BiometricEvaluation::MPI::Resources:



## Public Member Functions

- [Resources](#) (const std::string &propertiesFileName)  
*Constructor taking the name of the properties file describing the resources.*
- std::string [getPropertiesFileName](#) () const  
*Obtain the name of the file used to construct this object.*
- std::string [getLogsheetsURL](#) () const  
*Obtain the Uniform Resource Locator for the [IO:Logsheets](#) object.*
- int [getRank](#) () const
- int [getNumTasks](#) () const
- int [getWorkersPerNode](#) () const

## Static Public Member Functions

- static std::vector< std::string > [getRequiredProperties](#) ()  
*Obtain the list of required properties.*
- static std::vector< std::string > [getOptionalProperties](#) ()  
*Obtain the list of optional properties.*

## Static Public Attributes

- static const std::string [WORKERSPERNODEPROPERTY](#)  
The property string “Workers Per Node”; required.
- static const std::string [LOGSHEETURLPROPERTY](#)  
The property string “Logsheets URL”; optional.

### F.98.1 Detailed Description

A class to represent a set of resources needed by an [MPI](#) program. The resources are based on a properties file as well as some dynamic information, such as [MPI](#) rank and process ID.

### F.98.2 Constructor & Destructor Documentation

**BiometricEvaluation::MPI::Resources::Resources ( const std::string & *propertiesFileName* )**

Constructor taking the name of the properties file describing the resources.

Parameters

in	<i>propertiesFile↵ Name</i>	The name of the file containing the Properties.
----	---------------------------------	---

Exceptions

<a href="#">Error::FileError</a>	The resources file could not be read.
<a href="#">Error::ObjectDoesNot↵ Exist</a>	A required property does not exist.
<a href="#">Error::Exception</a>	Some other error occurred.

### F.98.3 Member Function Documentation

**std::string BiometricEvaluation::MPI::Resources::getLogsheetsURL ( ) const**

Obtain the Uniform Resource Locator for the [IO](#):Logsheets object.

This string may be empty, indicating that there is no Logsheets URL in the Properties file.

Returns

The Logsheets URL.

**static std::vector<std::string> BiometricEvaluation::MPI::Resources::getOptionalProperties ( )  
[static]**

Obtain the list of optional properties.

Returns

A set of optional property strings.

**std::string BiometricEvaluation::MPI::Resources::getPropertiesFileName ( ) const**

Obtain the name of the file used to construct this object.

Returns

The name of the properties file.

```
static std::vector<std::string> BiometricEvaluation::MPI::Resources::getRequiredProperties ( )
[static]
```

Obtain the list of required properties.

Returns

A set of required property strings.

## F.99 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](#) extraction\_method, int index\_one, int index\_two, int count=0)

Create a [RidgeCountItem](#) struct.

### Public Attributes

- [RidgeCountExtractionMethod](#) extraction\_method
- int index\_one
- int index\_two
- int count

#### F.99.1 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.

## F.100 BiometricEvaluation::MPI::Runtime Class Reference

[Runtime](#) support for the startup/shutdown of [MPI](#) jobs.

```
#include <be_mpi_runtime.h>
```

### Public Member Functions

- [Runtime](#) (int &argc, char \*\*&argv)
 

Construct the runtime environment for the processes making up the [MPI](#) job.
- void start ([BiometricEvaluation::MPI::Distributor](#) &distributor, [BiometricEvaluation::MPI::Receiver](#) &receiver)
 

Startup the runtime environment for the [MPI](#) job.
- void shutdown ()
 

Shutdown the runtime environment for the [MPI](#) job.
- void abort (int errcode)
 

Abort the runtime the [MPI](#) job.

### F.100.1 Detailed Description

[Runtime](#) support for the startup/shutdown of [MPI](#) jobs.

This class provides methods that are used by applications to start and shutdown the [MPI](#) job. Each job consists of a single distributor of work, and 1..n receivers of work which then distribute the work packages to child processes to take action on the work package.

### F.100.2 Constructor & Destructor Documentation

**BiometricEvaluation::MPI::Runtime::Runtime ( int &argc, char \*\*&argv )**

Construct the runtime environment for the processes making up the [MPI](#) job.

Parameters

in	<i>argc</i>	The argument count, taken from the command line passed to main().
in	<i>argv</i>	The argument vector, taken from the command line passed to main().

### F.100.3 Member Function Documentation

**void BiometricEvaluation::MPI::Runtime::abort ( int *errcode* )**

Abort the runtime the [MPI](#) job.

This method will cause the [MPI](#) job to terminate immediately. All processes will end without the opportunity to save.

Parameters

in	<i>errcode</i>	The error code to return to the <a href="#">MPI</a> runtime.
----	----------------	--

**void BiometricEvaluation::MPI::Runtime::shutdown ( )**

Shutdown the runtime environment for the [MPI](#) job.

This method must be called in order for the [MPI](#) runtime to cleanly exit.

**void BiometricEvaluation::MPI::Runtime::start ( BiometricEvaluation::MPI::Distributor & distributor, BiometricEvaluation::MPI::Receiver & receiver )**

Startup the runtime environment for the [MPI](#) job.

Parameters

in	<i>distributor</i>	The <a href="#">Distributor</a> object that will form the basis of the first <a href="#">MPI</a> task.
in	<i>receiver</i>	The <a href="#">Receiver</a> object which will form the basis of <a href="#">MPI</a> tasks 1..n.

## F.101 BiometricEvaluation::Process::Semaphore Class Reference

Represent a semaphore that can be used for interprocess communication.

```
#include <be-process-semaphore.h>
```

### Public Member Functions

- [Semaphore](#) (const std::string &name, const mode\_t mode, const int value, const bool exclusive=false)  
*Create a new named sempahore.*
- [Semaphore](#) (const std::string &name)

- Open an existing named semaphore.*
- bool [wait](#) (const bool interruptible)
- Wait indefinitely for the semaphore to unblock.*
- bool [trywait](#) (const bool interruptible)
- Attempt to obtain the semaphore without blocking.*
- bool [timedwait](#) (const uint64\_t interval, const bool interruptible)
- Attempt to obtain the semaphore while blocking for at most the specified time interval.*
- void [post](#) ()
- Post (increment) to the semaphore.*

### F.101.1 Detailed Description

Represent a semaphore that can be used for interprocess communication.

Semaphores are shared counters with mutually exclusive modification properties. A counter value greater than zero means that a resource represented by the semaphore is available. A typical use is to grant exclusive access to a resource by allowing the counter to be valued at zero or one; this is known as a binary semaphore.

Note

The counter value is not exposed to clients of the object.

### F.101.2 Constructor & Destructor Documentation

**BiometricEvaluation::Process::Semaphore::Semaphore ( const std::string & *name*, const mode\_t *mode*, const int *value*, const bool *exclusive* = *false* )**

Create a new named semaphore.

Parameters

in	<i>name</i>	The name of the semaphore, which must obey the syntax documented for the sem_open(2) call. If the semaphore already exists in the name space, construction will fail unless the exclusive flag is true. In that case, the existing semaphore will be removed.
in	<i>mode</i>	The permission mode of the semaphore.
in	<i>value</i>	The initial value of the semaphore.
in	<i>exclusive</i>	The semaphore is created only when it doesn't already exist.

Exceptions

<a href="#">Error::ObjectExists</a>	The semaphore already exists with the given name.
<a href="#">Error::StrategyError</a>	An error occurred when creating the semaphore.

**BiometricEvaluation::Process::Semaphore::Semaphore ( const std::string & *name* )**

Open an existing named semaphore.

Parameters

in	<i>name</i>	The name of the semaphore, which must obey the syntax documented for the sem_open(2) call.
----	-------------	--

### F.101.3 Member Function Documentation

**void BiometricEvaluation::Process::Semaphore::post ( )**

Post (increment) to the semaphore.

Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The semaphore is no longer valid.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">System</a> error obtaining the semaphore.

**bool BiometricEvaluation::Process::Semaphore::timedwait ( const uint64\_t *interval*, const bool *interruptible* )**

Attempt to obtain the semaphore while blocking for at most the specified time interval.

Parameters

<i>in</i>	<i>interval</i>	The max time to wait, in microseconds.
<i>in</i>	<i>interruptible</i>	true if the function should return if waiting was interrupted, false otherwise.

Returns

true if the semaphore was obtained; false if not.

Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The semaphore is no longer valid.
<a href="#"><i>Error::NotImplemented</i></a>	Function is not implemented on the system. Applications should then call <a href="#">wait()</a> or <a href="#">trywait()</a> .
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">System</a> error obtaining the semaphore.

**bool BiometricEvaluation::Process::Semaphore::trywait ( const bool *interruptible* )**

Attempt to obtain the semaphore without blocking.

Parameters

<i>in</i>	<i>interruptible</i>	true if the function should return if waiting was interrupted, false otherwise.
-----------	----------------------	---

Returns

true if the semaphore was obtained; false if not.

Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The semaphore is no longer valid.
<a href="#"><i>Error::StrategyError</i></a>	<a href="#">System</a> error obtaining the semaphore.

**bool BiometricEvaluation::Process::Semaphore::wait ( const bool *interruptible* )**

Wait indefinitely for the semaphore to unblock.

Parameters

in	interruptible	true if the function should return if waiting was interrupted, false otherwise.
----	---------------	---

Returns

true if the semaphore was obtained; false if not.

Error::ObjectDoesNotExist	The semaphore is no longer valid.
Error::StrategyError	System error obtaining the semaphore.

## F.102 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](#) ()
- [SignalManager](#) (const sigset\_t signalSet)
- void [setSignalSet](#) (const sigset\_t signalSet)
- void [clearSignalSet](#) ()
- void [setDefaultSignalSet](#) ()
- bool [sigHandled](#) ()
- void [start](#) ()
- void [stop](#) ()
- void [setSigHandled](#) ()
- void [clearSigHandled](#) ()

### Static Public Attributes

- static bool [\\_canSigJump](#)
- static sigjmp\_buf [\\_sigJumpBuf](#)

#### F.102.1 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.

The ABORT\_SIGNAL\_MANAGER() macro also disables the watchdog timer but does not create the code point destination for the jump point. This macro should be used to disable a [SignalManager](#) object when the application is no longer interested in the signal handling.



## Attention

The `BEGIN_SIGNAL_BLOCK()` macro must be paired with either the `END_SIGNAL_BLOCK()` macro or `ABORT_SIGNAL_MANAGER()` macro. Failure to do so may result in undefined behavior as an active [SignalManager](#) may be invoked, forcing a jump into an incompletely initialized function.

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.

## F.102.2 Constructor & Destructor Documentation

### **BiometricEvaluation::Error::SignalManager::SignalManager ( )**

Construct a new [SignalManager](#) object with the default signal handling: SIGSEGV and SIGBUS.  
Exceptions

<a href="#">Error::StrategyError</a>	Could not register the signal handler.
--------------------------------------	--

### **BiometricEvaluation::Error::SignalManager::SignalManager ( const sigset\_t signalSet )**

Construct a new [SignalManager](#) object with the specified signal handling, no defaults.

Parameters

<i>signalSet</i>	(in) The signal set; see <code>sigaction(2)</code> , <code>sigemptyset(3)</code> and <code>sigaddset(3)</code> .
------------------	--

Exceptions

<a href="#">Error::ParameterError</a>	One of the signals in <code>signalSet</code> cannot be handled (SIGKILL, SIGSTOP).
---------------------------------------	--

## F.102.3 Member Function Documentation

### **void BiometricEvaluation::Error::SignalManager::clearSigHandled ( )**

Clear the indication that a signal was handled.

### **void BiometricEvaluation::Error::SignalManager::clearSignalSet ( )**

Clear all signal handling.

### **void BiometricEvaluation::Error::SignalManager::setDefaultSignalSet ( )**

Set the default signals this object will manage: SIGSEGV and SIGBUS.

### **void BiometricEvaluation::Error::SignalManager::setSigHandled ( )**

Set a flag to indicate a signal was handled.

### **void BiometricEvaluation::Error::SignalManager::setSignalSet ( const sigset\_t signalSet )**

Set the signals this object will manage.

## Parameters

<i>signalSet</i>	(in) The signal set; see sigaction(2), sigemptyset(3) and sigaddset(3).
------------------	---

## Exceptions

<a href="#"><i>Error::ParameterError</i></a>	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP).
--	---

**bool BiometricEvaluation::Error::SignalManager::sigHandled ( )**

Indicate whether a signal was handled.

## Returns

true if a signal was handled, false otherwise.

**void BiometricEvaluation::Error::SignalManager::start ( )**

Start handling signals of the current signal set.

## Exceptions

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

**void BiometricEvaluation::Error::SignalManager::stop ( )**

Stop handling signals of the current signal set.

## Exceptions

<a href="#"><i>Error::StrategyError</i></a>	Could not register the signal handler.
---	--

**F.102.4 Member Data Documentation****bool BiometricEvaluation::Error::SignalManager::canSigJump [static]**

Flag indicating can jump after handling a signal.

## Note

Should not be directly used by applications.

**sigjmp\_buf BiometricEvaluation::Error::SignalManager::sigJumpBuf [static]**

The jump buffer used by the signal handler.

## Note

Should not be directly used by applications.

## F.103 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](#)

### F.103.1 Detailed Description

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

### F.103.2 Constructor & Destructor Documentation

**BiometricEvaluation::Image::Size::Size** ( const uint32\_t *xSize* = 0, const uint32\_t *ySize* = 0 )

Create a [Size](#) struct.

Parameters

in	<i>xSize</i>	Number of pixels on the X-axis
in	<i>ySize</i>	Number of pixels on the Y-axis

### F.103.3 Member Data Documentation

**uint32\_t BiometricEvaluation::Image::Size::xSize**

Number of pixels on the X-axis

**uint32\_t BiometricEvaluation::Image::Size::ySize**

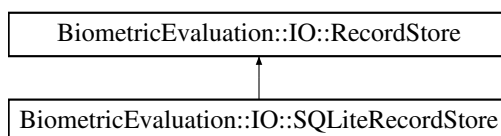
Number of pixels on the Y-axis

## F.104 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 std::string &pathname, const std::string &description)
- **SQLiteRecordStore** (const std::string &pathname, uint8\_t mode=READWRITE)
- void **move** (const std::string &pathname)  
*Move the [RecordStore](#).*
- void **changeDescription** (const std::string &description)
- uint64\_t **getSpaceUsed** () const  
*Obtain real storage utilization.*
- void **insert** (const std::string &key, const void \*const data, const uint64\_t size)
- void **remove** (const std::string &key)
- uint64\_t **read** (const std::string &key, void \*const data) const
- void **replace** (const std::string &key, const void \*const data, const uint64\_t size)
- uint64\_t **length** (const std::string &key) const
- void **flush** (const std::string &key) const
- uint64\_t **sequence** (std::string &key, void \*const data=nullptr, int cursor=BE\_RECSTORE\_SEQ\_NEXT)  
*Sequence through a [RecordStore](#), returning the key/data pairs.*
- void **setCursorAtKey** (const std::string &key)
- **SQLiteRecordStore** (const [SQLiteRecordStore](#) &)=delete
- [SQLiteRecordStore](#) & **operator=** (const [SQLiteRecordStore](#) &)=delete

## Protected Member Functions

- void **sqliteError** (int32\_t errorNumber) const  
*Convert an SQLite error into a StrategyError.*
- void **createStructure** ()  
*Create the tables needed to store key->value pairs in SQLite.*
- bool **validateKeyValueType** (const std::string &table)  
*Confirm that a key->value table exists with the proper schema.*
- void **createKeyValueType** (const std::string &table)  
*Create a tables needed to store key->value pairs in SQLite.*
- bool **validateSchema** ()  
*Confirm that the schema of the opened SQLite database is compatible.*
- uint64\_t **readSegments** (const std::string &key, void \*const data) const  
*Select a row from the [RecordStore](#).*
- void **cleanup** ()  
*Perform SQLite cleanup routines.*

## Additional Inherited Members

### F.104.1 Detailed Description

A [RecordStore](#) implementation using a SQLite database as the underlying record storage system.

### F.104.2 Member Function Documentation

**void BiometricEvaluation::IO::SQLiteRecordStore::changeDescription ( const std::string &description ) [virtual]**

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

Parameters

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

Exceptions

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

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

**void BiometricEvaluation::IO::SQLiteRecordStore::cleanup ( ) [protected]**

Perform SQLite cleanup routines.

- Finalize the sequencer statement
- Close the SQLite database handle

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	Bad return code from SQLite during cleanup.
---	---

**void BiometricEvaluation::IO::SQLiteRecordStore::createKeyValueTable ( const std::string & *table* ) [protected]**

Create a tables needed to store key->value pairs in SQLite.

Parameters

<i>table</i>	Name of the table to create.
--------------	------------------------------

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> executing SQL commands.
---	---

**void BiometricEvaluation::IO::SQLiteRecordStore::createStructure ( ) [protected]**

Create the tables needed to store key->value pairs in SQLite.

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	<a href="#">Error</a> executing SQL commands.
---	---

**void BiometricEvaluation::IO::SQLiteRecordStore::flush ( const std::string & *key* ) const [virtual]**

Commit the record's data to storage.

Parameters

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

Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	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](#).

**uint64\_t BiometricEvaluation::IO::SQLiteRecordStore::getSpaceUsed ( ) const [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](#).

**void BiometricEvaluation::IO::SQLiteRecordStore::insert ( const std::string & key, const void \*const data, const uint64\_t size ) [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](#).

**uint64\_t BiometricEvaluation::IO::SQLiteRecordStore::length ( const std::string & key ) const [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](#).

```
void BiometricEvaluation::IO::SQLiteRecordStore::move ( const std::string & pathname )  
[virtual]
```

Move the [RecordStore](#).

The [RecordStore](#) can be moved to a new path in the file system.

## Parameters

in	<i>pathname</i>	The new path of the <a href="#">RecordStore</a> .
----	-----------------	---

## Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::SQLiteRecordStore::read ( const std::string & *key*, void \*const *data* ) const** [**virtual**]

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

## Parameters

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

## Returns

The size of the record.

## Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::SQLiteRecordStore::readSegments ( const std::string & *key*, void \*const *data* ) const** [**protected**]

Select a row from the [RecordStore](#).

## Parameters

<i>key</i>	Key of the row to select.
<i>data</i>	If not nullptr, deep copy the record for key into data.

## Exceptions

<a href="#">Error::ObjectDoesNotExist</a>	Key does not exist in <a href="#">RecordStore</a> .
<a href="#">Error::StrategyError</a>	Error executing SQL commands.

## Returns

Size of key's record.

**void BiometricEvaluation::IO::SQLiteRecordStore::remove ( const std::string & *key* )** [**virtual**]

Remove a record from the store.



## Parameters

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

## Exceptions

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

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

**void BiometricEvaluation::IO::SQLiteRecordStore::replace ( const std::string & key, const void \*const data, const uint64\_t size ) [virtual]**

Replace a complete record in a store.

## Parameters

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

## Exceptions

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

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

**uint64\_t BiometricEvaluation::IO::SQLiteRecordStore::sequence ( std::string & key, void \*const data = nullptr, int cursor = BE\_RECSTORE\_SEQ\_NEXT ) [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 nullptr to indicate only the key is wanted.
in	cursor	The location within the sequence of the key/data pair to return.

## Returns

The length of the record currently in sequence.

## Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	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](#).

**void BiometricEvaluation::IO::SQLiteRecordStore::setCursorAtKey ( const std::string & key )**  
**[virtual]**

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

Parameters

in	key	The key of the record which will be returned by the first subsequent call to <a href="#">sequence()</a> .
----	-----	---

Exceptions

<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](#).

**void BiometricEvaluation::IO::SQLiteRecordStore::sqliteError ( int32\_t errorNumber ) const**  
**[protected]**

Convert an SQLite error into a StrategyError.

Exceptions

<i>Error::StrategyError</i>	Always thrown with the textual description of the last error condition.
-----------------------------	---

**bool BiometricEvaluation::IO::SQLiteRecordStore::validateKeyValueTable ( const std::string & table )**  
**[protected]**

Confirm that a key->value table exists with the proper schema.

Parameters

table	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.
-----------------------------	----------------------

**bool BiometricEvaluation::IO::SQLiteRecordStore::validateSchema ( )**  
**[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

<a href="#">Error::StrategyError</a>	Error compiling SQL.
--------------------------------------	----------------------

## F.105 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::FileLogCabinet](#) \*const logCabinet)
- void [getCPUTimes](#) (uint64\_t \*usertime, uint64\_t \*systemtime)
- void [getMemorySizes](#) (uint64\_t \*vmrss, uint64\_t \*vmsize, uint64\_t \*vmpeak, uint64\_t \*vmdata, uint64\_t \*vmstack)
- uint32\_t [getNumThreads](#) ()
- void [logStats](#) ()
  - Create a snapshot of the current process statistics in the FileLogsheets created in the FileLogCabinet.*
- void [startAutoLogging](#) (uint64\_t interval)
  - 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](#) ()
  - Stop the automatic logging of process statistics.*
- void [callStatistics\\_logStats](#) ()

### F.105.1 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 FileLogsheets object contained within the provided FileLogCabinet.

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.

### F.105.2 Constructor & Destructor Documentation

**BiometricEvaluation::Process::Statistics::Statistics ( )**

Constructor with no parameters.

**BiometricEvaluation::Process::Statistics::Statistics ( [IO::FileLogCabinet](#) \*const logCabinet )**

Construct a [Statistics](#) object with the associated FileLogCabinet.

Parameters

in	<i>logCabinet</i>	The FileLogCabinet object where this object will create a FileLogsheet to contain the statistic information for the process.
----	-------------------	--

Exceptions

<i>Error::NotImplemented</i>	Logging is not supported on this OS. This exception can be thrown when any portion of the statistics gathering cannot be completed.
<i>Error::ObjectExists</i>	The FileLogsheet already exists. This exception should rarely, if ever, occur.
<i>Error::StrategyError</i>	Failure to create the FileLogsheet in the cabinet.

### F.105.3 Member Function Documentation

**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.

**void BiometricEvaluation::Process::Statistics::getCPUTimes ( uint64\_t \* *usertime*, uint64\_t \* *systemtime* )**

Obtain the total user and system times for the process, in microseconds. Any of the out parameters can be nullptr, 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.

**void BiometricEvaluation::Process::Statistics::getMemorySizes ( uint64\_t \* *vmrss*, uint64\_t \* *vmsize*, uint64\_t \* *vmpeak*, uint64\_t \* *vmdata*, uint64\_t \* *vmstack* )**

Obtain the current memory set sizes for the process, in kilobytes. Any of the out parameters can be nullptr, 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

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when obtaining the process statistics from the operating system. The exception information string contains the error reason.
<a href="#"><i>Error::NotImplemented</i></a>	This method is not implemented on this OS.

### **uint32\_t BiometricEvaluation::Process::Statistics::getNumThreads ( )**

Obtain the number of threads composing this process.

Note

This method may not be implemented in all operating systems.

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when obtaining the process info from the operating system. The exception information string contains the error reason.
<a href="#"><i>Error::NotImplemented</i></a>	This method is not implemented on this OS.

### **void BiometricEvaluation::Process::Statistics::logStats ( )**

Create a snapshot of the current process statistics in the FileLogsheet created in the FileLogCabinet.

Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The FileLogsheet does not exist; this object was not created with FileLogCabinet object.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when writing to the FileLogsheet.
<a href="#"><i>Error::NotImplemented</i></a>	The statistics gathering is not implemented for this operating system.

### **void BiometricEvaluation::Process::Statistics::startAutoLogging ( uint64\_t interval )**

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

<code>in</code>	<i>interval</i>	The gap between logging snapshots, in microseconds.
-----------------	-----------------	---

Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	The FileLogsheet does not exist; this object was not created with FileLogCabinet object.
<a href="#"><i>Error::ObjectExists</i></a>	Autologging is currently invoked.
<a href="#"><i>Error::StrategyError</i></a>	An error occurred when writing to the FileLogsheet.
<a href="#"><i>Error::NotImplemented</i></a>	The statistics gathering is not implemented for this operating system.

### **void BiometricEvaluation::Process::Statistics::stopAutoLogging ( )**

Stop the automatic logging of process statistics.

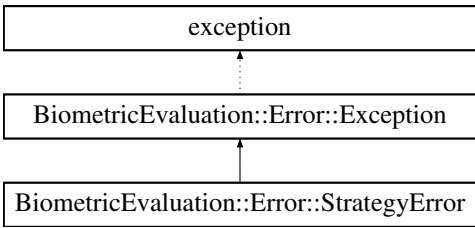
Exceptions

<a href="#">Error::ObjectDoesNotExist</a>	Not currently autologging.
<a href="#">Error::StrategyError</a>	An error occurred when stopping, most likely because the logging thread died.

F.106 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](#) (std::string info)

F.106.1 Detailed Description

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

F.106.2 Constructor & Destructor Documentation

**BiometricEvaluation::Error::StrategyError::StrategyError ( )**

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

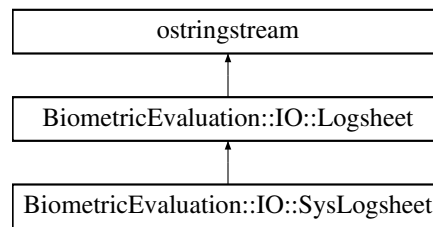
**BiometricEvaluation::Error::StrategyError::StrategyError ( std::string info )**

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

F.107 BiometricEvaluation::IO::SysLogsheet Class Reference

A class to represent a single logging mechanism to a logging service on the network.

```
#include <be_io_syslogsheet.h>
Inheritance diagram for BiometricEvaluation::IO::SysLogsheet:
```



## Public Member Functions

- [SysLogsheet](#) (const std::string &url, const std::string &description, const std::string &appname, bool sequenced, bool utc)

*Create a new log sheet.*

- [SysLogsheet](#) (const std::string &url, const std::string &description, const std::string &appname, const std::string &hostname, bool sequenced, bool utc)

*Create a new log sheet.*

- [~SysLogsheet](#) ()
- void [write](#) (const std::string &entry)

*Write a string as an entry to the backing store.*

- void [writeComment](#) (const std::string &entry)

*Write a string as a comment to the backing store.*

- void [writeDebug](#) (const std::string &entry)

*Write a string as a debug entry to the backing store.*

- void [sync](#) ()

*Synchronize any buffered data to the underlying backing store.*

## Protected Member Functions

- [SysLogsheet](#) (const [SysLogsheet](#) &)
- [SysLogsheet](#) & [operator=](#) (const [SysLogsheet](#) &)
- void [setup](#) (const std::string &url, const std::string &description)
- void [writeToLogger](#) (const std::string &priority, const char delimiter, const std::string &prefix, const std::string &message)

## Protected Attributes

- std::string [\\_hostname](#)
- std::string [\\_appname](#)
- std::string [\\_procid](#)
- int [\\_sockFD](#)
- bool [\\_sequenced](#)
- bool [\\_operational](#)
- bool [\\_utc](#)

## Additional Inherited Members

### F.107.1 Detailed Description

A class to represent a single logging mechanism to a logging service on the network.

Log entries are sent to the logging server in RFC5424 format with a timestamp of the local system in UTC. Normal and comment entries are sent to the logger with a PRI field indicating the 'local0' facility and a severity of 'Informational'. Debug entries are sent with facility of 'local1' and severity 'Debug'. A basic syslog config file would contain these lines: local0.info /var/log/info.log local1.debug /var/log/debug.log

The hostname is added to each entry but may be overridden by constructing the object with a given hostname, including the RFC5424 NILVALUE character. The PROCID part of each log message will be filled in with the process ID. Multi-line messages are segmented and sent to the logger as separate entries with the same timestamp and sequence number.

### F.107.2 Constructor & Destructor Documentation

**BiometricEvaluation::IO::SysLogsheet::SysLogsheet ( const std::string & url, const std::string & description, const std::string & appname, bool sequenced, bool utc )**

Create a new log sheet.

Parameters

in	<i>url</i>	The Uniform Resource Locator describing the logging service. Accepted forms are syslog://hostname:port
in	<i>description</i>	The text used to describe the sheet. This text is written into the log prior to any entries.
in	<i>appname</i>	The name of the application. This text is written into each log entry.
in	<i>sequenced</i>	True if each entry should include a sequence number, false if not.
in	<i>utc</i>	True if timestamps should be in Coordinated Universal <a href="#">Time</a> (UTC), false for local time.

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when connecting to the logging system, or URL is malformed.
---	---

**BiometricEvaluation::IO::SysLogsheet::SysLogsheet ( const std::string & url, const std::string & description, const std::string & appname, const std::string & hostname, bool sequenced, bool utc )**

Create a new log sheet.

Parameters

in	<i>url</i>	The Uniform Resource Locator describing the logging service. Accepted forms are syslog://hostname:port
in	<i>description</i>	The text used to describe the sheet. This text is written into the log prior to any entries.
in	<i>appname</i>	The name of the application. This text is written into each log entry.
in	<i>hostname</i>	The string to use as the hostname for all log entries.
in	<i>sequenced</i>	True if each entry should include a sequence number, false if not.
in	<i>utc</i>	True if timestamps should be in Coordinated Universal <a href="#">Time</a> (UTC), false for local time.



Exceptions

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when connecting to the logging system, or URL is malformed.
---	---

**BiometricEvaluation::IO::SysLogsheet::~~SysLogsheet ( )**

Destructor

**BiometricEvaluation::IO::SysLogsheet::SysLogsheet ( const SysLogsheet & ) [protected]**

Prevent copying of [SysLogsheet](#) objects

### F.107.3 Member Function Documentation

**SysLogsheet& BiometricEvaluation::IO::SysLogsheet::operator= ( const SysLogsheet & ) [protected]**

Prevent copying of [SysLogsheet](#) objects

**void BiometricEvaluation::IO::SysLogsheet::setup ( const std::string & *url*, const std::string & *description* ) [protected]**

Helper function to build connections

**void BiometricEvaluation::IO::SysLogsheet::sync ( ) [virtual]**

Synchronize any buffered data to the underlying backing store.

This syncing is dependent on the behavior of the underlying storage mechanism.

Exceptions

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

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

**void BiometricEvaluation::IO::SysLogsheet::write ( const std::string & *entry* ) [virtual]**

Write a string as an entry to the backing store.

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

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

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

**void BiometricEvaluation::IO::SysLogsheet::writeComment ( const std::string & *entry* ) [virtual]**

Write a string as a comment to the backing store.

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

<code>in</code>	<code>entry</code>	The text of the comment.
-----------------	--------------------	--------------------------

## Exceptions

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

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

**void BiometricEvaluation::IO::SysLogsheet::writeDebug ( const std::string & *entry* ) [virtual]**

Write a string as a debug entry to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A debug line is prefixed with DebugDelimiter followed by a space.

## Parameters

<code>in</code>	<code>entry</code>	The text of the debug message.
-----------------	--------------------	--------------------------------

## Exceptions

<a href="#"><i>Error::StrategyError</i></a>	An error occurred when logging.
---	---------------------------------

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

**void BiometricEvaluation::IO::SysLogsheet::writeToLogger ( const std::string & *priority*, const char *delimiter*, const std::string & *prefix*, const std::string & *message* ) [protected]**

Helper function to write to the logger

## F.107.4 Member Data Documentation

**bool BiometricEvaluation::IO::SysLogsheet::operational [protected]**

Whether the sheet is operational

**bool BiometricEvaluation::IO::SysLogsheet::sequenced [protected]**

Whether to include entry sequence numbers

**int BiometricEvaluation::IO::SysLogsheet::sockFD [protected]**

Socket file descriptor for the logging system

**bool BiometricEvaluation::IO::SysLogsheet::utc [protected]**

Whether time stamps are in UTC

## F.108 BiometricEvaluation::MPI::TaskCommand Class Reference

The command given to an [MPI](#) task.

```
#include <be_mpi.h>
```

## Public Types

- enum [Kind](#) {  
**Continue** = 0, **Ignore** = 1, **Exit** = 2, **QuickExit** = 3,  
**TermExit** = 4 }

### F.108.1 Detailed Description

The command given to an [MPI](#) task.

### F.108.2 Member Enumeration Documentation

**enum BiometricEvaluation::MPI::TaskCommand::Kind**

Enumerator

- Ignore** Normal operation.
- Exit** Ignore the message.
- QuickExit** Transition to the normal shutdown state.
- TermExit** Transition to the quick shutdown state.

## F.109 BiometricEvaluation::MPI::TaskStatus Class Reference

The status of an [MPI](#) distributor or receiver task.

```
#include <be_mpi.h>
```

## Public Types

- enum [Kind](#) { **OK** = 0, **Failed** = 1, **Exit** = 2 }

### F.109.1 Detailed Description

The status of an [MPI](#) distributor or receiver task.

### F.109.2 Member Enumeration Documentation

**enum BiometricEvaluation::MPI::TaskStatus::Kind**

Enumerator

- Failed** Normal operation.
- Exit** Failed to complete an operation.

## F.110 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 Types

- using [BE\\_CLOCK\\_TYPE](#) = std::chrono::steady\_clock

## Public Member Functions

- [Timer](#) ()
- void [start](#) ()  
*Start tracking time.*
- void [stop](#) ()  
*Stop tracking time.*
- uint64\_t [elapsed](#) () const  
*Get the elapsed time in microseconds between calls to this object's [start\(\)](#) and [stop\(\)](#) methods.*

### F.110.1 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.

#### Warning

Timers are not threadsafe and should only be used to time operations within the same thread.

### F.110.2 Member Typedef Documentation

**using BiometricEvaluation::Time::Timer::BE\_CLOCK\_TYPE = std::chrono::steady\_clock**

Clock type to use, aliased for easy replacement.

### F.110.3 Constructor & Destructor Documentation

**BiometricEvaluation::Time::Timer::Timer ( )**

Constructor for the [Timer](#) object.

### F.110.4 Member Function Documentation

**uint64\_t BiometricEvaluation::Time::Timer::elapsed ( ) const**

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

<a href="#">Error::StrategyError</a>	This object is currently timing an operation or an error occurred when obtaining timing information.
--------------------------------------	--

**void BiometricEvaluation::Time::Timer::start ( )**

Start tracking time.

Exceptions

<a href="#"><i>Error::StrategyError</i></a>	This object is currently timing an operation or an error occurred when obtaining timing information.
---	--

**void BiometricEvaluation::Time::Timer::stop ( )**

Stop tracking time.

Exceptions

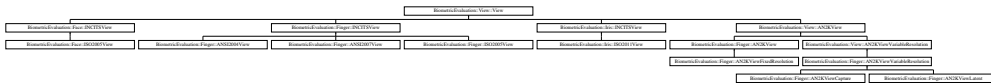
<a href="#"><i>Error::StrategyError</i></a>	This object is not currently timing an operation or an error occurred when obtaining timing information.
---	--

## F.111 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

- `std::shared_ptr< Image::Image > getImage () const`  
Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- `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 getCompressionAlgorithm () const`  
Obtain the compression algorithm used on the image.
- `Image::Resolution getScanResolution () const`  
Obtain the image scan resolution.

### Protected Member Functions

- `void setImageSize (const BiometricEvaluation::Image::Size &imageSize)`  
Mutator for the image size.
- `void setImageDepth (uint32_t imageDepth)`  
Mutator for the image size.
- `void setImageResolution (const BiometricEvaluation::Image::Resolution &imageResolution)`  
Mutator for the image resolution.
- `void setScanResolution (const BiometricEvaluation::Image::Resolution &scanResolution)`

- *Mutator for the image scan resolution.*
- void [setImageData](#) (const [BiometricEvaluation::Memory::uint8Array](#) &imageData)  
*Mutator for the image data.*
- void [setCompressionAlgorithm](#) (const [Image::CompressionAlgorithm](#) &ca)  
*Mutator for the compression algorithm.*

### F.111.1 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.

### F.111.2 Member Function Documentation

**[Image::CompressionAlgorithm](#) [BiometricEvaluation::View::View::getCompressionAlgorithm](#) ( ) const**

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.

Returns

The compression algorithm.

**[std::shared\\_ptr<Image::Image>](#) [BiometricEvaluation::View::View::getImage](#) ( ) const**

Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)

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

Returns

The image data.

**[uint32\\_t](#) [BiometricEvaluation::View::View::getImageDepth](#) ( ) const**

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.

Returns

The image depth.

**[Image::Resolution](#) [BiometricEvaluation::View::View::getImageResolution](#) ( ) const**

Obtain the image resolution.

[Image](#) resolution is taken from the biometric record, and not from the image data.

Returns

The scan resolution.

Note

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.

**Image::Size BiometricEvaluation::View::View::getImageSize ( ) const**

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.

Returns

The image size.

**Image::Resolution BiometricEvaluation::View::View::getScanResolution ( ) const**

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.

Returns

The scan resolution.

Note

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.

**void BiometricEvaluation::View::View::setImageData ( const BiometricEvaluation::Memory::uint8↵ Array & *imageData* ) [protected]**

Mutator for the image data.

Parameters

in	<i>imageData</i>	The image data object.
----	------------------	------------------------

**void BiometricEvaluation::View::View::setImageDepth ( uint32\_t *imageDepth* ) [protected]**

Mutator for the image size.

Parameters

in	<i>imageDepth</i>	The image depth.
----	-------------------	------------------

**void BiometricEvaluation::View::View::setImageResolution ( const BiometricEvaluation::Image::↵ Resolution & *imageResolution* ) [protected]**

Mutator for the image resolution.

Parameters

in	<i>image↵ Resolution</i>	The image resolution object.
----	------------------------------	------------------------------

**void BiometricEvaluation::View::View::setImageSize ( const BiometricEvaluation::Image::Size & *imageSize* ) [protected]**

Mutator for the image size.

Parameters

<code>in</code>	<code>imageSize</code>	The image size object.
-----------------	------------------------	------------------------

**void BiometricEvaluation::View::View::setScanResolution ( const BiometricEvaluation::Image::Resolution & *scanResolution* ) [protected]**

Mutator for the image scan resolution.

Parameters

<code>in</code>	<code>scanResolution</code>	The image scan resolution object.
-----------------	-----------------------------	-----------------------------------

## F.112 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)
- void [setInterval](#) (uint64\_t interval)
- void [start](#) ()
- void [stop](#) ()
- 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](#)

### F.112.1 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 assigned interval value. Applications can set the interval once and use the block macros repeatedly. Failure to call `setInterval()` results in an effectively disabled timer, as does setting the interval to 0.

The `ABORT_WATCHDOG()` macro also disables the watchdog timer but does not create the code point destination for the jump point. This macro should be used to disable a [Watchdog](#) object when the application is no longer interested in the timeout condition.

#### Attention

The `BEGIN_WATCHDOG_BLOCK()` macro must be paired with either the `END_WATCHDOG_BLOCK()` macro or `ABORT_WATCHDOG_BLOCK()` macro. Failure to do so may result in undefined behavior as a running [Watchdog](#) timer may expire, forcing a jump into an incompletely initialized function.

#### 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](#)

## F.112.2 Constructor & Destructor Documentation

**BiometricEvaluation::Time::Watchdog::Watchdog ( `const uint8_t type` )**

Construct a new [Watchdog](#) object.

Parameters

<code>in</code>	<i>type</i>	The type of timer, <code>ProcessTime</code> or <code>RealTime</code> .
-----------------	-------------	--

Exceptions

<a href="#">Error::NotImplemented</a>	The type of watchdog requested is not implemented.
<a href="#">Error::ParameterError</a>	The type is invalid.

#### Warning

[Watchdog::PROCESSTIME](#) is not supported under Cygwin.

## F.112.3 Member Function Documentation

**void BiometricEvaluation::Time::Watchdog::clearCanSigJump ( )**

Clears the flag for the [Watchdog](#) object to indicate that the signal jump block is no longer valid.

**void BiometricEvaluation::Time::Watchdog::clearExpired ( )**

Clear the flag indicating the timer expired.

**bool BiometricEvaluation::Time::Watchdog::expired ( )**

Indicate whether the watchdog timer expired.

Returns

true if the timer expired, false otherwise.

**void BiometricEvaluation::Time::Watchdog::setCanSigJump ( )**

Indicate that the signal handler can jump into the application code after handling the signal.

**void BiometricEvaluation::Time::Watchdog::setExpired ( )**

Set a flag to indicate the timer expired.

**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.
-----------	-----------------	--------------------------------------

**void BiometricEvaluation::Time::Watchdog::start ( )**

Start a watchdog timer.

Exceptions

<a href="#">Error::StrategyError</a>	Could not register the signal handler, or could not create the timer.
--------------------------------------	---

**void BiometricEvaluation::Time::Watchdog::stop ( )**

Stop a watchdog timer.

Exceptions

<a href="#">Error::StrategyError</a>	Could not clear the timer.
--------------------------------------	----------------------------

## F.112.4 Member Data Documentation

**const uint8\_t BiometricEvaluation::Time::Watchdog::PROCESSTIME = 0 [static]**

A [Watchdog](#) based on process time.

**const uint8\_t BiometricEvaluation::Time::Watchdog::REALTIME = 1 [static]**

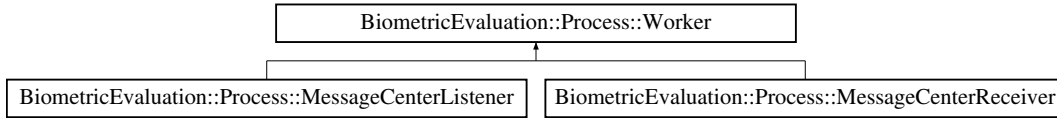
A [Watchdog](#) based on real (wall clock) time.

## F.113 BiometricEvaluation::Process::Worker Class Reference

An abstraction of an instance that performs work on given data.

```
#include <be_process_worker.h>
```

Inheritance diagram for BiometricEvaluation::Process::Worker:



### Public Member Functions

- virtual int32\_t [workerMain](#) ()=0  
*The method that will get called to start execution by a [ProcessManager](#).*
- std::shared\_ptr< void > [getParameter](#) (const std::string &name)  
*Obtain a parameter passed to this [Worker](#).*
- double [getParameterAsDouble](#) (const std::string &name)  
*Obtain a parameter passed to this [Worker](#) as a double.*
- int64\_t [getParameterAsInteger](#) (const std::string &name)  
*Obtain a parameter passed to this [Worker](#) as an integer.*
- std::string [getParameterAsString](#) (const std::string &name)  
*Obtain a parameter passed to this [Worker](#) as a string.*
- void [setParameter](#) (const std::string &name, std::shared\_ptr< void > argument)  
*Pass a parameter to this [Worker](#).*
- void [stop](#) ()  
*Tell this [Worker](#) to return ASAP.*
- void [closeWorkerPipeEnds](#) ()  
*Perform initialization for communication from [Worker](#) to [Manager](#).*
- void [closeManagerPipeEnds](#) ()  
*Perform initialization for communication from [Manager](#) to [Worker](#).*
- int [getSendingPipe](#) () const  
*Obtain the pipe used to send messages to this [Worker](#).*
- int [getReceivingPipe](#) () const  
*Obtain the pipe used to receive messages to this [Worker](#).*
- void [sendMessageToManager](#) (const [Memory::uint8Array](#) &message)  
*Send a message to the [Manager](#).*
- void [receiveMessageFromManager](#) ([Memory::uint8Array](#) &message)  
*Receive a message from the [Manager](#).*
- void [\\_initCommunication](#) ()  
*Perform general communication initialization from Constructor.*
- virtual [~Worker](#) ()  
*[Worker](#) destructor.*

## Protected Member Functions

- [Worker](#) ()  
*Worker constructor.*
- bool [stopRequested](#) () const  
*Determine if the parent has requested this child to exit.*
- bool [waitForMessage](#) (int numSeconds=-1) const  
*Block while waiting for a message from the [Manager](#).*

### F.113.1 Detailed Description

An abstraction of an instance that performs work on given data.

### F.113.2 Member Function Documentation

**void BiometricEvaluation::Process::Worker::initCommunication ( )**

Perform general communication initialization from Constructor.

Exceptions

<a href="#">Error::StrategyError</a>	<a href="#">Error</a> in initialization.
--------------------------------------	--

**void BiometricEvaluation::Process::Worker::closeManagerPipeEnds ( )**

Perform initialization for communication from [Manager](#) to [Worker](#).

Note

Behavior is undefined if called by a non-Worker.

Exceptions

<a href="#">Error::StrategyError</a>	Communications not enabled.
--------------------------------------	-----------------------------

**void BiometricEvaluation::Process::Worker::closeWorkerPipeEnds ( )**

Perform initialization for communication from [Worker](#) to [Manager](#).

Note

Behavior is undefined if called by a non-Manager.

Exceptions

<a href="#">Error::StrategyError</a>	Communications not enabled.
--------------------------------------	-----------------------------

**std::shared\_ptr<void> BiometricEvaluation::Process::Worker::getParameter ( const std::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.

## Exceptions

<i>std::out_of_range</i>	name was not set.
--------------------------	-------------------

**double BiometricEvaluation::Process::Worker::getParameterAsDouble ( const std::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.

## Exceptions

<i>std::out_of_range</i>	name was not set.
--------------------------	-------------------

**int64\_t BiometricEvaluation::Process::Worker::getParameterAsInteger ( const std::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.

## Exceptions

<i>std::out_of_range</i>	name was not set.
--------------------------	-------------------

**std::string BiometricEvaluation::Process::Worker::getParameterAsString ( const std::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.

Exceptions

<i>std::out_of_range</i>	name was not set.
--------------------------	-------------------

**int BiometricEvaluation::Process::Worker::getReceivingPipe ( ) const**

Obtain the pipe used to receive messages to this [Worker](#).

Returns

Receiving pipe.

Exceptions

<i>Error::ObjectDoesNotExist</i>	<a href="#">Worker</a> exiting soon, communication disabled.
<i>Error::StrategyError</i>	Communications not enabled.

**int BiometricEvaluation::Process::Worker::getSendingPipe ( ) const**

Obtain the pipe used to send messages to this [Worker](#).

Returns

Sending pipe.

Exceptions

<i>Error::ObjectDoesNotExist</i>	<a href="#">Worker</a> exiting soon, communication disabled.
<i>Error::StrategyError</i>	Communications not enabled.

**void BiometricEvaluation::Process::Worker::receiveMessageFromManager ( [Memory::uint8Array](#) & *message* )**

Receive a message from the [Manager](#).

Parameters

out	<i>message</i>	Buffer to store the received message.
-----	----------------	---------------------------------------

Exceptions

<i>Error::ObjectDoesNotExist</i>	Widowed pipe.
<i>Error::StrategyError</i>	Communications not enabled.

See also

[waitForMessage](#)

**void BiometricEvaluation::Process::Worker::sendMessageToManager ( const [Memory::uint8Array](#) & *message* )**

Send a message to the [Manager](#).

## Parameters

<i>in</i>	<i>message</i>	Message to send.
-----------	----------------	------------------

## Exceptions

<a href="#"><i>Error::ObjectDoesNotExist</i></a>	Widowed pipe.
<a href="#"><i>Error::StrategyError</i></a>	Communications not enabled.

**void BiometricEvaluation::Process::Worker::setParameter ( const std::string & *name*,  
std::shared\_ptr< void > *argument* )**

Pass a parameter to this [Worker](#).

## Parameters

<i>name</i>	A unique identifier for this parameter
<i>argument</i>	A shared_ptr to the object to store.

**void BiometricEvaluation::Process::Worker::stop ( )**

Tell this [Worker](#) to return ASAP.

## Attention

This method should not be overridden.

**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.

**bool BiometricEvaluation::Process::Worker::waitForMessage ( int *numSeconds* = -1 ) const  
[protected]**

Block while waiting for a message from the [Manager](#).

## Parameters

<i>numSeconds</i>	Number of seconds to wait for a message, or any value < 0 to wait forever.
-------------------	--

## Returns

true once a message is ready to be read or false if an error occurred.

**virtual int32\_t BiometricEvaluation::Process::Worker::workerMain ( ) [pure virtual]**

The method that will get called 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.

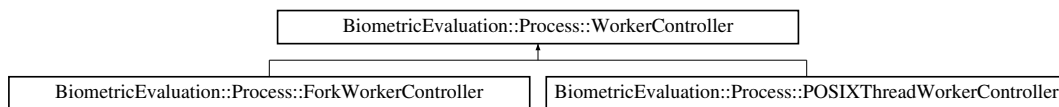
Implemented in [BiometricEvaluation::Process::MessageCenterReceiver](#), and [BiometricEvaluation::Process::MessageCenterListener](#).

## F.114 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](#) (std::shared\_ptr< [Worker](#) > worker)
- virtual void [sendMessageToWorker](#) (const [Memory::uint8Array](#) &message)
 

*Send a message to the [Worker](#) contained within this [WorkerController](#).*
- virtual void [setParameter](#) (const std::string &name, std::shared\_ptr< void > argument)
 

*Set the parameter to be passed to the [Worker](#).*
- virtual void [setParameterFromDouble](#) (const std::string &name, double argument)
 

*Set a double parameter to be passed to the [Worker](#).*
- virtual void [setParameterFromInteger](#) (const std::string &name, int64\_t argument)
 

*Set an integer parameter to be passed to the [Worker](#).*
- virtual void [setParameterFromString](#) (const std::string &name, const std::string &argument)
 

*Set a string parameter to be passed to the [Worker](#).*
- virtual void [reset](#) ()
 

*Reuse the [Worker](#).*
- virtual bool [isWorking](#) () const =0
 

*Obtain whether or not [Worker](#) is working.*
- virtual bool [everWorked](#) () const =0
 

*Obtain whether or not this [Worker](#) has ever worked.*
- bool [finishedWorking](#) () const
 

*Obtain whether or not this [Worker](#) has both started and finished its task.*
- std::shared\_ptr< [Worker](#) > [getWorker](#) () const
 

*Obtain the [Worker](#) instance being wrapped.*
- virtual [~WorkerController](#) ()
 

*[WorkerController](#) destructor.*



## Protected Attributes

- `std::shared_ptr< Worker > _worker`

### F.114.1 Detailed Description

Wrapper of a [Worker](#) returned from a [Process::Manager](#).

### F.114.2 Constructor & Destructor Documentation

**BiometricEvaluation::Process::WorkerController::WorkerController** ( `std::shared_ptr< Worker > worker` )

[WorkerController](#) constructor.

Parameters

<i>worker</i>	The <a href="#">Worker</a> instance to wrap.
---------------	--

### F.114.3 Member Function Documentation

**virtual bool BiometricEvaluation::Process::WorkerController::everWorked** ( ) const **[pure virtual]**

Obtain whether or not this [Worker](#) has ever worked.

Returns

true the [Worker](#) has ever or is currently working, false otherwise.

Note

[reset\(\)](#) will change the result of this method.

Implemented in [BiometricEvaluation::Process::ForkWorkerController](#), and [BiometricEvaluation::Process::POSIXThreadWorkerController](#).

**bool BiometricEvaluation::Process::WorkerController::finishedWorking** ( ) const **[inline]**

Obtain whether or not this [Worker](#) has both started and finished its task.

Returns

true if the [Worker](#) has both started and finished performing its task, false otherwise.

Note

[reset\(\)](#) will change the result of this method.

**std::shared\_ptr<Worker> BiometricEvaluation::Process::WorkerController::getWorker** ( ) const

Obtain the [Worker](#) instance being wrapped.

Returns

[Worker](#) instance.

**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::POSIXThreadWorkerController](#).

**virtual void BiometricEvaluation::Process::WorkerController::reset ( ) [virtual]**

Reuse the [Worker](#).

Exceptions

<a href="#">Error::ObjectExists</a>	The previously started <a href="#">Worker</a> is still running.
-------------------------------------	---

Reimplemented in [BiometricEvaluation::Process::ForkWorkerController](#), and [BiometricEvaluation::Process::POSIXThreadWorkerController](#).

**virtual void BiometricEvaluation::Process::WorkerController::sendMessageToWorker ( const Memory::uint8Array & message ) [virtual]**

Send a message to the [Worker](#) contained within this [WorkerController](#).

Parameters

<i>message</i>	Message to send to the <a href="#">Worker</a> .
----------------	---

Exceptions

<a href="#">Error::ObjectDoesNotExist</a>	<a href="#">Worker</a> receive pipe is closed ( <a href="#">Worker</a> object likely destroyed).
<a href="#">Error::StrategyError</a>	Message sending failed.

**virtual void BiometricEvaluation::Process::WorkerController::setParameter ( const std::string & name, std::shared\_ptr< void > argument ) [virtual]**

Set the parameter to be passed to the [Worker](#).

Parameters

in	<i>name</i>	The name representing the argument in the <a href="#">Worker</a> .
in	<i>argument</i>	The argument to be passed to the <a href="#">Worker</a> .

Note

Subsequent calls to [setParameter\(\)](#) with the same name will overwrite any exiting argument.

**virtual void BiometricEvaluation::Process::WorkerController::setParameterFromDouble ( const std::string & name, double argument ) [virtual]**

Set a double parameter to be passed to the [Worker](#).

## Parameters

in	<i>name</i>	The name representing the argument in the <a href="#">Worker</a> .
in	<i>argument</i>	The double to be passed to the <a href="#">Worker</a> .

## Note

Subsequent calls to `setParameter*()` with the same name will overwrite any exiting argument.

**virtual void BiometricEvaluation::Process::WorkerController::setParameterFromInteger ( const std::string & name, int64\_t argument ) [virtual]**

Set an integer parameter to be passed to the [Worker](#).

## Parameters

in	<i>name</i>	The name representing the argument in the <a href="#">Worker</a> .
in	<i>argument</i>	The integer to be passed to the <a href="#">Worker</a> .

## Note

Subsequent calls to `setParameter*()` with the same name will overwrite any exiting argument.

**virtual void BiometricEvaluation::Process::WorkerController::setParameterFromString ( const std::string & name, const std::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 <a href="#">Worker</a> .
in	<i>argument</i>	The string to be passed to the <a href="#">Worker</a> .

## Note

Subsequent calls to `setParameter*()` with the same name will overwrite any exiting argument.

## F.114.4 Member Data Documentation

**std::shared\_ptr<Worker> BiometricEvaluation::Process::WorkerController::\_worker**  
[protected]

The [Worker](#) instance that is running in this child

## F.115 BiometricEvaluation::MPI::WorkPackage Class Reference

A class to represent a piece of work to be acted upon by a processor.

```
#include <be_mpi_workpackage.h>
```

### Public Member Functions

- [WorkPackage](#) ()  
*Construct an empty work package.*
- [WorkPackage](#) (const [Memory::uint8Array](#) &data)

- *Construct a work package with some data.*  
void [getData](#) ([Memory::uint8Array](#) &data) const
- *Obtain the package data in raw form.*  
void [setData](#) (const [Memory::uint8Array](#) &data)
- *Set the package data from raw data.*  
uint64\_t [getSize](#) () const
- *Obtain the size of the package data.*  
uint64\_t [getNumElements](#) () const
- *Obtain the number of elements in the package.*  
void [setNumElements](#) (const uint64\_t numElements)
- *Set the number of elements in the package.*

### F.115.1 Detailed Description

A class to represent a piece of work to be acted upon by a processor.

The work package is an wrapper around the data to be processed, along with some ancillary information.

### F.115.2 Constructor & Destructor Documentation

**BiometricEvaluation::MPI::WorkPackage::WorkPackage ( const [Memory::uint8Array](#) &data )**

Construct a work package with some data.

Parameters

<a href="#">in</a>	<a href="#">data</a>	The data that will be managed by this work package.
--------------------	----------------------	---

### F.115.3 Member Function Documentation

**uint64\_t BiometricEvaluation::MPI::WorkPackage::getNumElements ( ) const**

Obtain the number of elements in the package.

This value is determined by the application and must be set therein, otherwise 0 is returned.

Returns

The number of application defined elements in the work package.

**uint64\_t BiometricEvaluation::MPI::WorkPackage::getSize ( ) const**

Obtain the size of the package data.

Returns

The size (in octets) of the raw data item.

**void BiometricEvaluation::MPI::WorkPackage::setData ( const [Memory::uint8Array](#) &data )**

Set the package data from raw data.

Parameters

<code>in</code>	<code>data</code>	The data copied into the work package.
-----------------	-------------------	--

**void BiometricEvaluation::MPI::WorkPackage::setNumElements ( const uint64\_t numElements )**

Set the number of elements in the package.

Parameters

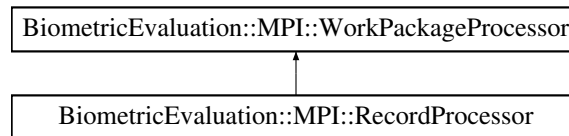
<code>in</code>	<code>numElements</code>	The number of application-defined elements in the work package.
-----------------	--------------------------	---

## F.116 BiometricEvaluation::MPI::WorkPackageProcessor Class Reference

Represents an object that processes the contents of a work package.

```
#include <be_mpi_workpackageprocessor.h>
```

Inheritance diagram for BiometricEvaluation::MPI::WorkPackageProcessor:



### Public Member Functions

- virtual std::shared\_ptr  
< [WorkPackageProcessor](#) > [newProcessor](#) (std::shared\_ptr< [IO::Logsheet](#) > &logsheet)=0  
*Obtain an object that will process work packages. This method is part of the factory personality.*
- virtual void [performInitialization](#) (std::shared\_ptr< [IO::Logsheet](#) > &logsheet)=0  
*Initialization function to be called before work is distributed to the work package processor.*
- virtual void [processWorkPackage](#) ([MPI::WorkPackage](#) &workPackage)=0  
*Process the data contents of the work package. This method is part of the worker personality.*
- void [setLogsheet](#) (std::shared\_ptr< [IO::Logsheet](#) > &logsheet)  
*Set the [IO::Logsheet](#) object that can be used to save message for objects of this class.*
- std::shared\_ptr< [IO::Logsheet](#) > [getLogsheet](#) ()  
*Obtain the [IO::Logsheet](#) object that can be used to save message for objects of this class.*

### F.116.1 Detailed Description

Represents an object that processes the contents of a work package.

A [WorkPackageProcessor](#) presents two personalities: One that of a worker to process work packages, and one that is a factory to return worker objects of the implementation class.

Subclasses of this class implement the functionality needed to perform an action on the work package data. The processing done by the implementation is application and data type specific.

Ultimately, the final implementation of the [WorkPackageProcessor](#) class is done in the application. Access to the Logsheet object maintained by the framework is provided by this class.

## F.116.2 Member Function Documentation

**std::shared\_ptr<IO::Logsheet> BiometricEvaluation::MPI::WorkPackageProcessor::getLogsheet ( )**

Obtain the [IO::Logsheet](#) object that can be used to save message for objects of this class.

Returns

logsheet A shared pointer to the Logsheet object.

**virtual std::shared\_ptr<WorkPackageProcessor> BiometricEvaluation::MPI::WorkPackageProcessor::newProcessor ( std::shared\_ptr< IO::Logsheet > & logsheet ) [pure virtual]**

Obtain an object that will process work packages. This method is part of the factory personality.

Parameters

<i>logsheet</i>	A shared pointer to the <a href="#">IO::Logsheet</a> that may be used to save messages generated by the object.
-----------------	---

Returns

A shared pointer to the work package processor.

Implemented in [BiometricEvaluation::MPI::RecordProcessor](#).

**virtual void BiometricEvaluation::MPI::WorkPackageProcessor::performInitialization ( std::shared\_ptr< IO::Logsheet > & logsheet ) [pure virtual]**

Initialization function to be called before work is distributed to the work package processor.

Implementations of this class can use this function to do any processing necessary before work is given to the processor, pre-forking.

This method is part of the factory personality. All state that is to be common across all package processor objects can be initialized in this method.

Parameters

<i>logsheet</i>	A shared pointer to the <a href="#">IO::Logsheet</a> that may be used to save messages generated by the object.
-----------------	---

Exceptions

<a href="#">Error::Exception</a>	An implementation specific. error occurred.
----------------------------------	---

Implemented in [BiometricEvaluation::MPI::RecordProcessor](#).

**virtual void BiometricEvaluation::MPI::WorkPackageProcessor::processWorkPackage ( MPI::WorkPackage & workPackage ) [pure virtual]**

[Process](#) the data contents of the work package. This method is part of the worker personality.

Parameters

<i>in</i>	<i>workPackage</i>	The work package.
-----------	--------------------	-------------------

## Exceptions

<a href="#"><i>Error::Exception</i></a>	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.
---	--

Implemented in [BiometricEvaluation::MPI::RecordProcessor](#).

**void BiometricEvaluation::MPI::WorkPackageProcessor::setLogsheet ( std::shared\_ptr< IO::Logsheet > &logsheet )**

Set the [IO::Logsheet](#) object that can be used to save message for objects of this class.

## Parameters

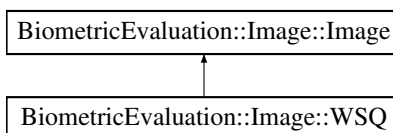
in	<i>logsheet</i>	A shared pointer to the Logsheet object.
----	-----------------	--

## F.117 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)
- [Memory::uint8Array getRawData](#) () const  
*Accessor for the raw image data. The data returned should not be compressed or encoded.*
- [Memory::uint8Array getRawGrayscaleData](#) (uint8\_t depth=8) const  
*Accessor for decompressed data in grayscale.*

### Static Public Member Functions

- static bool [isWSQ](#) (const uint8\_t \*data, uint64\_t size)

### Additional Inherited Members

#### F.117.1 Detailed Description

A WSQ-encoded image.

#### F.117.2 Member Function Documentation

**Memory::uint8Array BiometricEvaluation::Image::WSQ::getRawData ( ) const** **[virtual]**

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

## Returns

AutoArray holding raw image data.

## Exceptions

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

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

**Memory::uint8Array BiometricEvaluation::Image::WSQ::getRawGrayscaleData ( uint8\_t *depth* = 8 )  
const [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

AutoArray holding raw grayscale image data.

## Exceptions

<a href="#"><i>Error::DataError</i></a>	<a href="#">Error</a> decompressing image data.
<a href="#"><i>Error::ParameterError</i></a>	Invalid value for depth.

## Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

This method always returns an image that uses 8 bits to represent a single pixel. *depth* adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

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

**static bool BiometricEvaluation::Image::WSQ::isWSQ ( const uint8\_t \* *data*, uint64\_t *size* )  
[static]**

Whether or not data is a [WSQ](#) image.

## Parameters

<i>in</i>	<i>data</i>	The buffer to check.
<i>in</i>	<i>size</i>	The size of data.

## Returns

true if data appears to be a [WSQ](#) image, false otherwise



# Index

- Amputated
  - BiometricEvaluation::Finger::AN2KViewCapture, [138](#)
- Archive
  - BiometricEvaluation::IO::RecordStore, [354](#)
- Assisted
  - BiometricEvaluation::View::AN2KView, [135](#)
- Automatic
  - BiometricEvaluation::Feature::AN2K7Minutiae, [118](#)
- AutomaticEdited
  - BiometricEvaluation::Feature::AN2K7Minutiae, [118](#)
- AutomaticUnedited
  - BiometricEvaluation::Feature::AN2K7Minutiae, [118](#)
- Bandaged
  - BiometricEvaluation::Finger::AN2KViewCapture, [138](#)
- BerkeleyDB
  - BiometricEvaluation::IO::RecordStore, [354](#)
- BiometricEvaluation::Feature::AN2K7Minutiae
  - Automatic, [118](#)
  - AutomaticEdited, [118](#)
  - AutomaticUnedited, [118](#)
- BiometricEvaluation::Finger::AN2KViewCapture
  - Amputated, [138](#)
  - Bandaged, [138](#)
  - NA, [138](#)
- BiometricEvaluation::IO::Logsheet
  - File, [293](#)
  - Null, [293](#)
  - Syslog, [293](#)
- BiometricEvaluation::IO::RecordStore
  - Archive, [354](#)
  - BerkeleyDB, [354](#)
  - Compressed, [354](#)
  - Default, [354](#)
  - File, [354](#)
  - List, [354](#)
  - SQLite, [354](#)
- BiometricEvaluation::Image::Resolution
  - NA, [372](#)
  - PPCM, [372](#)
  - PPI, [372](#)
- BiometricEvaluation::MPI::MessageTag
  - Data, [311](#)
  - OOB, [311](#)
- BiometricEvaluation::MPI::TaskCommand
  - Exit, [398](#)
  - Ignore, [398](#)
  - QuickExit, [398](#)
  - TermExit, [398](#)
- BiometricEvaluation::MPI::TaskStatus
  - Exit, [399](#)
  - Failed, [399](#)
- BiometricEvaluation::View::AN2KView
  - Assisted, [135](#)
  - Controlled, [135](#)
  - NA, [135](#)
  - Observed, [135](#)
  - Unattended, [135](#)
  - Unknown, [135](#)
- Compressed
  - BiometricEvaluation::IO::RecordStore, [354](#)
- Controlled
  - BiometricEvaluation::View::AN2KView, [135](#)
- Data
  - BiometricEvaluation::MPI::MessageTag, [311](#)
- Default
  - BiometricEvaluation::IO::RecordStore, [354](#)
- Exit
  - BiometricEvaluation::MPI::TaskCommand, [398](#)
  - BiometricEvaluation::MPI::TaskStatus, [399](#)
- Failed
  - BiometricEvaluation::MPI::TaskStatus, [399](#)
- File
  - BiometricEvaluation::IO::Logsheet, [293](#)
  - BiometricEvaluation::IO::RecordStore, [354](#)
- Ignore
  - BiometricEvaluation::MPI::TaskCommand, [398](#)
- List

BiometricEvaluation::IO::RecordStore, [354](#)

## NA

BiometricEvaluation::Finger::AN2KViewCapture,  
[138](#)

BiometricEvaluation::Image::Resolution, [372](#)

BiometricEvaluation::View::AN2KView, [135](#)

## Null

BiometricEvaluation::IO::Logsheet, [293](#)

## OOB

BiometricEvaluation::MPI::MessageTag, [311](#)

## Observed

BiometricEvaluation::View::AN2KView, [135](#)

## PPCM

BiometricEvaluation::Image::Resolution, [372](#)

## PPI

BiometricEvaluation::Image::Resolution, [372](#)

## PPMM

BiometricEvaluation::Image::Resolution, [372](#)

## QuickExit

BiometricEvaluation::MPI::TaskCommand, [398](#)

## SQLite

BiometricEvaluation::IO::RecordStore, [354](#)

## Syslog

BiometricEvaluation::IO::Logsheet, [293](#)

## TermExit

BiometricEvaluation::MPI::TaskCommand, [398](#)

## Unattended

BiometricEvaluation::View::AN2KView, [135](#)

## Unknown

BiometricEvaluation::View::AN2KView, [135](#)