Nail to Nail Fingerprint Capture Challenge

Auto-generated API Documentation

February 17, 2017

Contents

1 Main Page

1.1 Overview

This is the API for participant-specific one-to-many template generation and template matching algorithms for Intelligence Advanced Research Projects Activity's (IARPA) 2017 Nail to Nail Fingerprint Capture Challenge. This API is based off the API used for Fingerprint Vendor Technology Evaluation (FpVTE) 2012, by the National Institute of Standards and Technology (NIST), released in the public domain.

1.2 Implementation

A pure-virtual (abstract) class called N2N::Interface has been created. Participants must implement all methods of N2N::Interface in a subclass, and submit this implementation as a shared library. The name of the library must follow the instructions in N2N::Interface::getIDs(). A test application will link against the submitted library, instantiate an instance of the implementation by calling N2N::Interface↩ ::getImplementation(), and perform various template generation and template matching operations.

1.3 Contact

Additional information regarding the Nail to Nail Fingerprint Capture Challenge can be obtained by emailing N2NChallenge@iarpa.gov. Additional information regarding this API and the associated software test can be obtained by emailing N2NChallenge@nist.gov.

1.4 License

2 Namespace Documentation

2.1 BiometricEvaluation::Finger Namespace Reference

Biometric information relating to finger images 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** }

  Pattern classification codes.
- 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, **Left↩RightThumbs** = 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**, **LiveScanNon↩OpticalContactRolled**, **LiveScanOpticalContactlessPlain**,
  LiveScanOpticalContactlessRolled, **LiveScanNonOpticalContactlessPlain**, **LiveScan↩NonOpticalContactlessRolled**, **Other**,
  Unknown }

  Finger and palm impression types.
- enum FingerImageCode {
  EJI = 0, **RolledTip**, **FullFingerRolled**, **FullFingerPlainLeft**,
  FullFingerPlainCenter, **FullFingerPlainRight**, **ProximalSegment**, **DistalSegment**,
  MedialSegment, **NA** }

  Joint and tip codes.

2.1.1 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.

2.1.2 Enumeration Type Documentation

2.1.2.1 FingerImageCode

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

Joint and tip codes.

2.1.2.2 Impression

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

Finger and palm impression types.

2.1.2.3 PatternClassification

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

Pattern classification codes.

2.1.2.4 Position

`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.

2.2 BiometricEvaluation::Image Namespace Reference

Classes and methods for manipulating images.

Classes

- class Image

  Represent attributes common to all images.
- class Raw

  An image with no encoding or compression.

2.2.1 Detailed Description

Classes and methods for manipulating images.

2.3 N2N Namespace Reference

Contains all the structures and functions used by the API.

Classes

- struct Candidate

  Object used to report a single candidate in a candidate list.
- struct FingerImage

  Fingerprint image and image attributes.
- class Interface

  The interface to the implementations.
- struct ReturnStatus

  Information about the completion status of a method.

Typedefs

- using FingerImage = struct FingerImage

  Convenience type for struct FingerImage.
- using Candidate = struct Candidate

  Convenience type for struct Candidate.
- using ReturnStatus = struct ReturnStatus

  Convenience definition of struct ReturnStatus.

Enumerations

- enum StatusCode {
  StatusCode::Success = 0, StatusCode::ImageSizeNotSupported = 1, StatusCode::TemplateType↩
  NotSupported = 2, StatusCode::FailedToExtract = 3,
  StatusCode::FailedToSearch = 4, StatusCode::FailedToParseInput = 5, StatusCode::Insufficient↩
  Resources = 6, StatusCode::Vendor = 7 }

  The status codes that are returned from a function call.
- enum InputType { InputType::Capture, InputType::Latent }

  Classes of imagery that can be provided.

Functions

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

  Output stream operator for a StatusCode object.
- std::ostream & operator<< (std::ostream &s, const ReturnStatus &rs)

  Output stream operator for a ReturnStatus object.

2.3.1 Detailed Description

Contains all the structures and functions used by the API.

2.3.2 Typedef Documentation

2.3.2.1 Candidate

```
using N2N::Candidate = typedef struct Candidate
```

Convenience type for struct Candidate.

2.3.2.2 FingerImage

```
using N2N::FingerImage = typedef struct FingerImage
```

Convenience type for struct FingerImage.

2.3.2.3 ReturnStatus

```
using N2N::ReturnStatus = typedef struct ReturnStatus
```

Convenience definition of struct ReturnStatus.

2.3.3 Enumeration Type Documentation

2.3.3.1 InputType

```
enum N2N::InputType  [strong]
```

Classes of imagery that can be provided.

Enumerator

Capture	Images where the subject was present during collection.
Latent	Images where a subject was not present during collection.

2.3.3.2 StatusCode

```
enum N2N::StatusCode  [strong]
```

The status codes that are returned from a function call.

Enumerator

Success	Successful completion.
ImageSizeNotSupported	Image size too small or large.
TemplateTypeNotSupported	Unsupported template type.

Enumerator

FailedToExtract	Could not extract template from image.
FailedToSearch	Could not search enrollment set.
FailedToParseInput	Failure to parse data.
InsufficientResources	There are not enough resources to complete the task.
Vendor	Vendor-defined error.

2.3.4 Function Documentation

2.3.4.1 operator<<() [1/2]

```
std::ostream& N2N::operator<< (
              std::ostream & s,
              const StatusCode & sc )  [inline]
```

Output stream operator for a StatusCode object.

2.3.4.2 operator<<() [2/2]

```
std::ostream& N2N::operator<< (
              std::ostream & s,
              const ReturnStatus & rs )  [inline]
```

Output stream operator for a ReturnStatus object.

3 Class Documentation

3.1 BiometricEvaluation::Memory::AutoArray< T > Class Template Reference

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

`#include <be_memory_autoarray.h>`

Public Types

- using value_type = T
  Type of element.
- using size_type = size_t
  Type of subscripts, counts, etc.
- using iterator = AutoArrayIterator< false, T >
  Iterator of element.
- using const_iterator = AutoArrayIterator< true, T >
  Const iterator of element.
- using reference = T &
  Reference to element.
- using const_reference = const T &
  Const reference element.

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.
- std::vector< T > to_vector () const
  Obtain a copy of elements in this AutoArray as a vector.
- AutoArray (size_type size=0)
  Construct an AutoArray.
- AutoArray (const AutoArray &copy)
  Construct an AutoArray.
- AutoArray (AutoArray &&rvalue) noexcept
  Construct an AutoArray.
- AutoArray (std::initializer_list< T > ilist)
  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 ()
  Destructor.

3.1.1 Detailed Description

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

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

Objects of this type should be treated in the traditional manner for containers, where (size_type) construction creates an array of the given size, while {...} construction creates an array with the given elements.

3.1.2 Member Typedef Documentation

3.1.2.1 size_type

```
template<class T>
using BiometricEvaluation::Memory::AutoArray< T >::size_type = size_t
```

Type of subscripts, counts, etc.

3.1.3 Constructor & Destructor Documentation

3.1.3.1 AutoArray() [1/4]

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T >::AutoArray (
            size_type size = 0 )  [explicit]
```

Construct an AutoArray.

Parameters

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

Exceptions

Error::MemoryError	Could not allocate new memory.

3.1.3.2 AutoArray() [2/4]

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

Construct an AutoArray.

Parameters

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

Exceptions

Error::MemoryError	Could not allocate new memory.

3.1.3.3 AutoArray() [3/4]

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

Construct an AutoArray.

Parameters

in	*rvalue*	An rvalue reference to an AutoArray whose contents will be moved and destroyed.

3.1.3.4 AutoArray() [4/4]

```
template<class T>
BiometricEvaluation::Memory::AutoArray< T >::AutoArray (
                std::initializer_list< T > ilist )
```

Construct an AutoArray.

Parameters

in	*ilist*	An initializer list of type T.

3.1.4 Member Function Documentation

3.1.4.1 at() [1/2]

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T >::reference BiometricEvaluation::Memory::AutoArray< T >::at (
                ptrdiff_t index )
```

Subscript into the AutoArray with checked access.

Parameters

in	*index*	Subscript into underlying storage.

Returns

Reference to the element at the specified index.

Exceptions

out_of_range	Specified index is outside the bounds of this AutoArray.

3.1.4.2 at() [2/2]

```
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

index	Subscript into underlying storage.

Returns

Const reference to the element at the specified index.

Exceptions

out_of_range	Specified index is outside the bounds of this AutoArray.

3.1.4.3 begin() [1/2]

```
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.

3.1.4.4 begin() [2/2]

```
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.

3.1.4.5 cbegin()

```
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.

3.1.4.6 cend()

```
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.

3.1.4.7 copy() [1/2]

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

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

Parameters

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

Warning

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

3.1.4.8 copy() [2/2]

```
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

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

Warning

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

3.1.4.9 end() [1/2]

```
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.

3.1.4.10 end() [2/2]

```
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.

3.1.4.11 operator const T *()

```
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.

3.1.4.12 operator T*()

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

Convert AutoArray to T array.

Returns

Pointer to the beginning of the underlying array storage.

3.1.4.13 operator=() [1/2]

```
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

in	*other*	AutoArray to be copied.

Returns

Reference to a new AutoArray object, the lvalue AutoArray.

Exceptions

Error::MemoryError	Could not allocate new memory.

3.1.4.14 operator=() [2/2]

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

Move assignment operator.

Parameters

in	*other*	rvalue reference to another AutoArray, whose contents will be moved and cleared from itself.

Returns

Reference to the lvalue AutoArray.

3.1.4.15 operator[]() [1/2]

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

Subscripting operator overload with unchecked access.

Parameters

in	*index*	Subscript into underlying storage.

Returns

Reference to the element at the specified index.

3.1.4.16 operator[]() [2/2]

```
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

in	*index*	Subscript into underlying storage.

Returns

Const reference to the element at the specified index.

3.1.4.17 resize()

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

Change the number of accessible elements.

Parameters

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

Exceptions

Error::MemoryError	Problem allocating memory.

3.1.4.18 size()

```
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.

3.1.4.19 to_vector()

```
template<class T >
std::vector< T > BiometricEvaluation::Memory::AutoArray< T >::to_vector ( ) const
```

Obtain a copy of elements in this AutoArray as a vector.

Warning

A key difference between vectors and AutoArrays is that all elements of a vector must be initialized. Calling this method on an AutoArray where not all elements have been initialized will likely cause undefined behavior.

Returns

A vector containing the contents of this AutoArray.

3.2 N2N::Candidate Struct Reference

Object used to report a single candidate in a candidate list.

`#include <n2n.h>`

Public Member Functions

- Candidate ()=default
  Constructor.
- Candidate (const std::string &templateID, double similarity)
  Constructor.

Public Attributes

- std::string templateID {}
  Candidate's ID, as provided during finalizeEnrollment().
- double similarity {-1}
  Score reflecting similarity between candidate represented by ***templateID*** *and search template.*

3.2.1 Detailed Description

Object used to report a single candidate in a candidate list.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Candidate() [1/2]

```
N2N::Candidate::Candidate ( )  [default]
```

Constructor.

3.2.2.2 Candidate() [2/2]

```
N2N::Candidate::Candidate (
            const std::string & templateID,
            double similarity )  [inline]
```

Constructor.

Parameters

`in`	*templateID*	Candidate ID, as provided during finalizeEnrollment().
`in`	*similarity*	Similarity of `templateID` to search template.

3.2.3 Member Data Documentation

3.2.3.1 templateID

```
std::string N2N::Candidate::templateID {}
```

Candidate's ID, as provided during finalizeEnrollment().

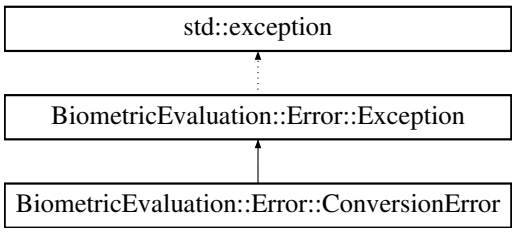
In a candidate list, the empty string represents that no candidate was found at this position.

3.3 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 ()

  Construct a ConversionError object with the default information string.
- ConversionError (const std::string &info)

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

3.3.1 Detailed Description

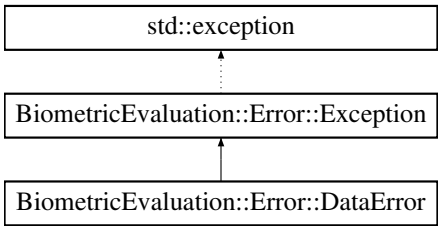
Error when converting one object into another, a property value from string to int, for example.

3.4 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 ()

  Construct a DataError object with the default information string.
- DataError (const std::string &info)

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

3.4.1 Detailed Description

Error when reading data from an external source.

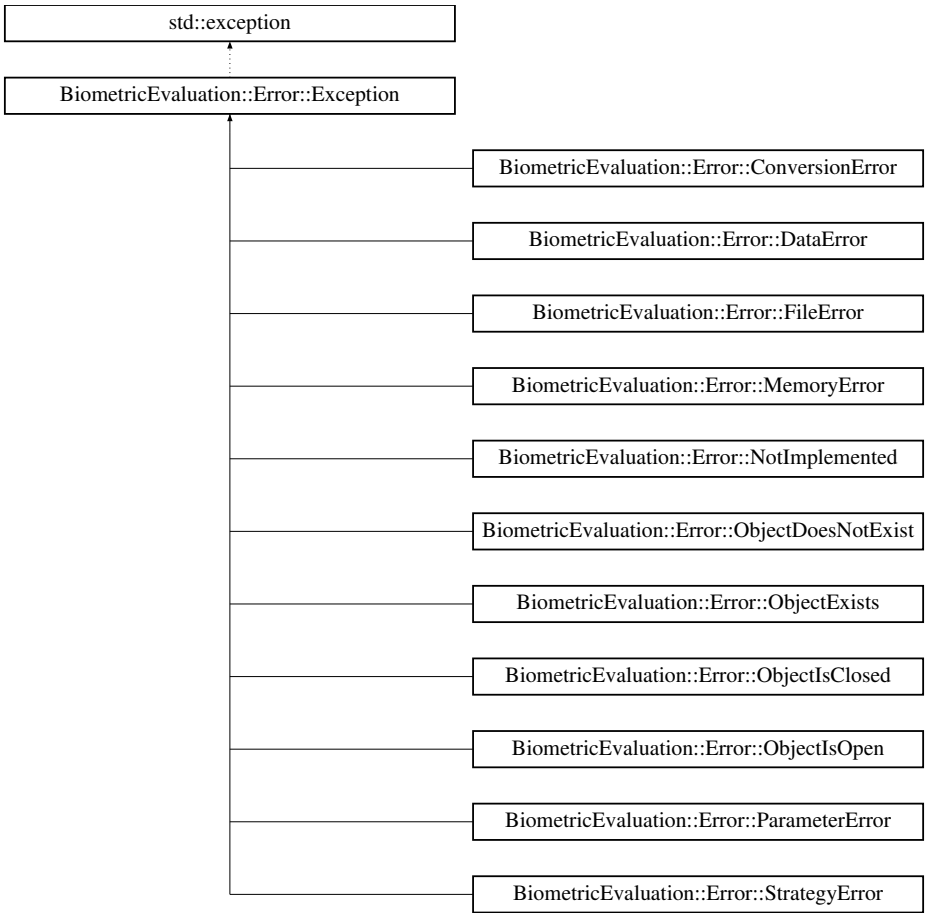
Typically occurs when reading data from a standard record, ANST/NIST 2000, for example, and a required field is missing, or a field has invalid data.

3.5 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 ()
  Construct an Exception object without an information string.
- Exception (std::string info)
  Construct an Exception object with an information string.
- const char * what () const noexcept
  Obtain the information string associated with the exception.
- const std::string whatString () const noexcept
  Obtain the information string associated with the exception.

3.5.1 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.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Exception()

```
BiometricEvaluation::Error::Exception::Exception (
            std::string info )
```

Construct an Exception object with an information string.

Parameters

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

3.5.3 Member Function Documentation

3.5.3.1 what()

```
const char* BiometricEvaluation::Error::Exception::what ( ) const  [noexcept]
```

Obtain the information string associated with the exception.

Returns

The information string as a char array.

3.5.3.2 whatString()

```
const std::string BiometricEvaluation::Error::Exception::whatString ( ) const  [noexcept]
```

Obtain the information string associated with the exception.

Returns

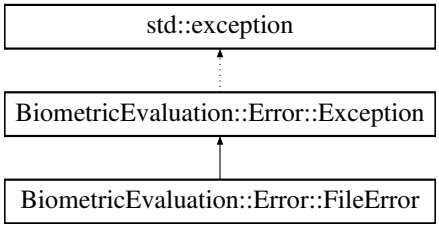
The information string.

3.6 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 ()
  Construct a FileError object with the default information string.
- FileError (const std::string &info)
  Construct a FileError object with an information string appended to the default information string.

3.6.1 Detailed Description

File error when opening, reading, writing, etc.

3.7 N2N::FingerImage Struct Reference

Fingerprint image and image attributes.

`#include <n2n.h>`

Public Member Functions

- FingerImage ()=default
  Constructor.
- FingerImage (const BiometricEvaluation::Finger::Position &fgp, const BiometricEvaluation::↩ Finger::Impression &imp, const uint8_t nfiq2, const std::shared_ptr< BiometricEvaluation::↩ Image::Raw > &rawImage)
  Constructor.

Public Attributes

- BiometricEvaluation::Finger::Position fgp
  Finger position of finger in ***rawImage****.*
- BiometricEvaluation::Finger::Impression imp
  Impression type of finger in ***rawImage****.*
- uint8_t nfiq2 {254}
  NFIQ2 value.
- std::shared_ptr< BiometricEvaluation::Image::Raw > rawImage {}
  Input image data, containing one finger.

3.7.1 Detailed Description

Fingerprint image and image attributes.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 FingerImage()

```
N2N::FingerImage::FingerImage (
            const BiometricEvaluation::Finger::Position & fgp,
            const BiometricEvaluation::Finger::Impression & imp,
            const uint8_t nfiq2,
            const std::shared_ptr< BiometricEvaluation::Image::Raw > & rawImage )  [inline]
```

Constructor.

Parameters

in	*fgp*	Finger position of finger in `rawImage`.
in	*imp*	Impression type of finger in `rawImage`.
in	*nfiq2*	NFIQ2 value of `rawImage`.
in	*rawImage*	Input image data.

3.7.3 Member Data Documentation

3.7.3.1 fgp

```
BiometricEvaluation::Finger::Position N2N::FingerImage::fgp
```

Initial value:

```
{
                BiometricEvaluation::Finger::Position::Unknown}
```

Finger position of finger in `rawImage`.

3.7.3.2 imp

```
BiometricEvaluation::Finger::Impression N2N::FingerImage::imp
```

Initial value:

```
{
                BiometricEvaluation::Finger::Impression::Unknown}
```

Impression type of finger in `rawImage`.

Note

No differentiation is provided between "traditional" and "participant sensor" rolled impressions.

3.7.3.3 nfiq2

```
uint8_t N2N::FingerImage::nfiq2 {254}
```

NFIQ2 value.

Meaning	Value
Quality	0 (low) – 100 (high)
Not Calculated	254
Error During Calculation	255

3.7.3.4 rawImage

`std::shared_ptr<BiometricEvaluation::Image::Raw> N2N::FingerImage::rawImage {}`

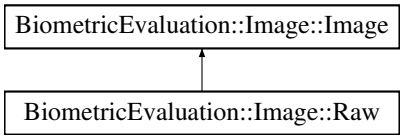
Input image data, containing one finger.

3.8 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 colorDepth, const uint16_t bitDepth, const Resolution resolution, const CompressionAlgorithm compression, const bool hasAlphaChannel)

  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.
- virtual Memory::uint8Array getRawData () const =0

  Accessor for the raw image data.
- virtual Memory::uint8Array getRawData (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data.
- virtual Memory::uint8Array getRawGrayscaleData (uint8_t depth) const =0

  Accessor for decompressed data in grayscale.
- Size getDimensions () const

  Accessor for the dimensions of the image in pixels.
- uint32_t getColorDepth () const

  Accessor for the color depth of the image in bits.
- uint16_t getBitDepth () const

  Accessor for the number of bits per color component.
- bool hasAlphaChannel () const

  Accessor for the presence of an alpha channel.

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 BiometricEvaluation::Image::Raw getRawImage (const std::shared_ptr< BiometricEvaluation::Image::Image > &image)

  Obtain Image::Raw version of an Image::Image.

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 setColorDepth (const uint32_t colorDepth)

  Mutator for the color depth of the image in bits.
- void setBitDepth (const uint16_t bitDepth)

  Mutator for the number of bits per component for color components in the image, in bits.
- const uint8_t * getDataPointer () const
- uint64_t getDataSize () const
- void setHasAlphaChannel (const bool hasAlphaChannel)

  Mutator for the presence of an alpha channel.

3.8.1 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.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Image() [1/2]

```
BiometricEvaluation::Image::Image::Image (
                const uint8_t * data,
                const uint64_t size,
                const Size dimensions,
                const uint32_t colorDepth,
                const uint16_t bitDepth,
                const Resolution resolution,
                const CompressionAlgorithm compression,
                const bool hasAlphaChannel )
```

Parent constructor for all Image classes.

Parameters

in	*data*	The image data.
in	*size*	The size of the image data, in bytes.
in	*dimensions*	The width and height of the image in pixels.
in	*colorDepth*	The image color depth, in bits-per-pixel.
in	*bitDepth*	The number of bits per color component.
in	*resolution*	The resolution of the image
in	*compression*	The CompressionAlgorithm of data.
in	*hasAlphaChannel*	Presence of an alpha channel.

Exceptions

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

3.8.2.2 Image() [2/2]

```
BiometricEvaluation::Image::Image::Image (
                const uint8_t * data,
                const uint64_t size,
                const CompressionAlgorithm compression )
```

Parent constructor for all Image classes.

Parameters

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

Exceptions

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

3.8.3 Member Function Documentation

3.8.3.1 getBitDepth()

```
uint16_t BiometricEvaluation::Image::Image::getBitDepth ( ) const
```

Accessor for the number of bits per color component.

Returns

The bit depth of the image (in bits).

3.8.3.2 getColorDepth()

```
uint32_t BiometricEvaluation::Image::Image::getColorDepth ( ) const
```

Accessor for the color depth of the image in bits.

Returns

The color depth of the image (bit).

3.8.3.3 getCompressionAlgorithm() [1/4]

```
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().

3.8.3.4 getCompressionAlgorithm() [2/4]

```
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	*data*	The image data.
in	*size*	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.

3.8.3.5 getCompressionAlgorithm() [3/4]

```
static CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm (
            const Memory::uint8Array & data )  [static]
```

Determine the compression algorithm of a buffer of image data.

Parameters

in	*data*	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.

3.8.3.6 getCompressionAlgorithm() [4/4]

```
static CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm (
            const std::string & path )  [static]
```

Determine the compression algorithm of a file.

Parameters

in	*path*	Path to file.

Returns

Compression algorithm used in the file.

Exceptions

Error::ObjectDoesNotExist	path does not exist.
Error::StrategyError	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.

3.8.3.7 getData()

```
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.

3.8.3.8 getDataPointer()

```
const uint8_t* BiometricEvaluation::Image::Image::getDataPointer ( ) const  [protected]
```

Returns

Const pointer to buffer underlying _data.

3.8.3.9 getDataSize()

```
uint64_t BiometricEvaluation::Image::Image::getDataSize ( ) const  [protected]
```

Returns

Size of _data.

3.8.3.10 getDimensions()

```
Size BiometricEvaluation::Image::Image::getDimensions ( ) const
```

Accessor for the dimensions of the image in pixels.

Returns

Coordinate object containing dimensions in pixels.

3.8.3.11 getRawData() [1/2]

```
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

Error::DataError	Error decompressing image data.

Implemented in BiometricEvaluation::Image::Raw.

3.8.3.12 getRawData() [2/2]

```
virtual Memory::uint8Array BiometricEvaluation::Image::Image::getRawData (
              const bool removeAlphaChannelIfPresent ) const  [virtual]
```

Accessor for the raw image data.

The data returned should not be compressed or encoded.

Parameters

in	*removeAlphaChannelIfPresent*	Whether or not to remove an alpha channel if one exists.

Returns

AutoArray holding raw image data, without an alpha channel if requested.

Exceptions

Error::DataError	Error decompressing image data.
Error::ParameterError	Propagated from Image::removeComponents.
Error::StrategyError	Propagated from Image::removeComponents.

3.8.3.13 getRawGrayscaleData()

```
virtual Memory::uint8Array BiometricEvaluation::Image::Image::getRawGrayscaleData (
              uint8_t depth ) const  [pure virtual]
```

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 16, 8, or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError	Error decompressing image data.
Error::NotImplemented	Unsupported conversion based on source color depth.
Error::ParameterError	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.
Alpha channels are completely ignored when converting to grayscale.

Implemented in BiometricEvaluation::Image::Raw.

3.8.3.14 getRawImage()

```
static BiometricEvaluation::Image::Raw BiometricEvaluation::Image::Image::getRawImage (
            const std::shared_ptr< BiometricEvaluation::Image::Image > & image )  [static]
```

Obtain Image::Raw version of an Image::Image.

Parameters

`in`	*image*	Shared pointer to an Image::Image.

Returns

Shared pointer to an Image::Raw version of `image`.

Note

If `image` is already an Image::Raw, `image` is returned to avoid a copy.

3.8.3.15 getResolution()

```
Resolution BiometricEvaluation::Image::Image::getResolution ( ) const
```

Accessor for the resolution of the image.

Returns

Resolution struct

3.8.3.16 hasAlphaChannel()

```
bool BiometricEvaluation::Image::Image::hasAlphaChannel ( ) const  [inline]
```

Accessor for the presence of an alpha channel.

Returns

Whether or not an alpha channel is present.

3.8.3.17 openImage() [1/3]

```
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	*data*	The image data.
in	*size*	The size of the image data, in bytes.

Returns

Image representation of the input data buffer.

Exceptions

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

3.8.3.18 openImage() [2/3]

```
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	*data*	The image data.

Returns

Image representation of the input data buffer.

Exceptions

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

3.8.3.19 openImage() [3/3]

```
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	*path*	Path to image data.

Returns

Image representation of the input data buffer.

Exceptions

Error::DataError	Error manipulating data.
Error::ObjectDoesNotExist	No file at specified path.
Error::StrategyError	Error while creating Image.

3.8.3.20 setBitDepth()

```
void BiometricEvaluation::Image::Image::setBitDepth (
                const uint16_t bitDepth )  [protected]
```

Mutator for the number of bits per component for color components in the image, in bits.

Parameters

in	*bitDepth*	The number of bits per color component.

3.8.3.21 setColorDepth()

```
void BiometricEvaluation::Image::Image::setColorDepth (
```

```
const uint32_t colorDepth )  [protected]
```

Mutator for the color depth of the image in bits.

Parameters

in	*colorDepth*	The color depth of the image (bit).

3.8.3.22 setDimensions()

```
void BiometricEvaluation::Image::Image::setDimensions (
        const Size dimensions )  [protected]
```

Mutator for the dimensions of the image in pixels.

Parameters

in	*dimensions*	Dimensions of image (pixel).

3.8.3.23 setHasAlphaChannel()

```
void BiometricEvaluation::Image::Image::setHasAlphaChannel (
        const bool hasAlphaChannel )  [inline], [protected]
```

Mutator for the presence of an alpha channel.

Parameters

in	*hasAlphaChannel*	Whether or not image has an alpha channel.

3.8.3.24 setResolution()

```
void BiometricEvaluation::Image::Image::setResolution (
        const Resolution resolution )  [protected]
```

Mutator for the resolution of the image .

Parameters

in	*resolution*	Resolution struct.

3.8.3.25 valueInColorspace()

```
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

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

Returns

A value equivalent to color in depth-bit space.

3.9 N2N::Interface Class Reference

The interface to the implementations.

```
#include <n2n.h>
```

Public Member Functions

- virtual void getIDs (std::string &identifier, uint32_t &revision, std::string &email)=0

  Obtain identifying information about the software under test.
- virtual ReturnStatus initMakeEnrollmentTemplate (const std::string &configurationDirectory)=0

  Prepare for calls to makeEnrollmentTemplate().
- virtual ReturnStatus makeEnrollmentTemplate (const std::vector< FingerImage > &standard↩ Images, const std::vector< BiometricEvaluation::Memory::uint8Array > &proprietaryImages, BiometricEvaluation::Memory::uint8Array &enrollmentTemplate)=0

  Create an enrollment template for one subject.
- virtual ReturnStatus finalizeEnrollment (const std::string &configurationDirectory, const std↩ ::string &enrollmentDirectory, const uint8_t nodeCount, const uint64_t nodeMemory, Biometric↩ Evaluation::IO::RecordStore &enrollmentTemplates)=0

  Form an enrollment set from one or more enrollment templates.
- virtual ReturnStatus initMakeSearchTemplate (const std::string &configurationDirectory, const InputType &inputType)=0

  Prepare for calls to makeSearchTemplate().
- virtual ReturnStatus makeSearchTemplate (const std::vector< FingerImage > &standardImages, const std::vector< BiometricEvaluation::Memory::uint8Array > &proprietaryImages, Biometric↩ Evaluation::Memory::uint8Array &searchTemplate)=0

  Create a search template for one subject.
- virtual ReturnStatus initIdentificationStageOne (const std::string &configurationDirectory, const std::string &enrollmentDirectory, const InputType &inputType, const uint8_t nodeNumber)=0

  Prepare for calls to identifyTemplateStageOne().
- virtual ReturnStatus identifyTemplateStageOne (const std::string &searchID, const Biometric↩ Evaluation::Memory::uint8Array &searchTemplate, const std::string &stageOneDataDirectory)=0

Search a template against the partial enrollment set.

- virtual ReturnStatus initIdentificationStageTwo (const std::string &configurationDirectory, const std::string &enrollmentDirectory, const InputType &inputType)=0

  Prepare for calls to identifyTemplateStageTwo().
- virtual ReturnStatus identifyTemplateStageTwo (const std::string &searchID, const std::string &stageOneDataDirectory, std::vector< Candidate > &candidates)=0

  Produce a candidate list from the results of all calls to identifyTemplateStageOne() for a particular search ID.
- virtual ~Interface ()=default

  Destructor.

Static Public Member Functions

- static std::shared_ptr< Interface > getImplementation ()

  Obtain a managed pointer to an implementation of this interface.

3.9.1 Detailed Description

The interface to the implementations.

The implementation under test will implement this interface by subclassing this class and implementing each method.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 ~Interface()

```
virtual N2N::Interface::~Interface ( )  [virtual], [default]
```

Destructor.

3.9.3 Member Function Documentation

3.9.3.1 finalizeEnrollment()

```
virtual ReturnStatus N2N::Interface::finalizeEnrollment (
               const std::string & configurationDirectory,
               const std::string & enrollmentDirectory,
               const uint8_t nodeCount,
               const uint64_t nodeMemory,
               BiometricEvaluation::IO::RecordStore & enrollmentTemplates )  [pure virtual]
```

Form an enrollment set from one or more enrollment templates.

This finalization step will prepare the enrolled templates to be distributed across multiple nodes. The enrollment directory will then be read-only throughout the duration of the identification process.

Parameters

in	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.
in	*enrollmentDirectory*	The top-level directory in which all enrollment data will reside. Access permission will be read-write and the application can populate this directory as needed. The directory is initially empty. After this method returns, the directory and its contents will become read-only.
in	*nodeCount*	The number of nodes the enrollment set will be spread across. It is up to the implementation to determine how best to spread the enrolled templates across the blades in order to get best performance. If `nodeCount` is not enough nodes, StatusCode::InsufficientResources should be returned.
in	*nodeMemory*	Amount of memory available to this process on each node, in kibibytes.
in	*enrollmentTemplates*	A read-only RecordStore of enrollment templates, as returned by makeEnrollmentTemplate().

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

All implementations must be capable of performing searches using >= 5 nodes. A larger value may be provided for speed, or a smaller value provided to conserve resources. If a smaller value is not feasible, StatusCode::InsufficientResources should be returned. Implementations that do not return successfully for values >= 5 will be disqualified.
The file system does not perform well with the creation of millions of small files, so the application should consolidate templates into some sort of database file within `enrollmentDirectory`.
This method must return within 90 minutes per 1-million subjects (e.g., if 5-million enrollment templates are provided, this method must return within 7.5 hours).
Reasonable multithreading is permitted. This method will only be called once.

3.9.3.2 getIDs()

```
virtual void N2N::Interface::getIDs (
                std::string & identifier,
                uint32_t & revision,
                std::string & email )  [pure virtual]
```

Obtain identifying information about the software under test.

Participants will receive an identifier from the project sponsor, and use this method to hard-code the identifier into the submission. The information obtained by this method must form the name of the submitted library, in the form `libN2N_<identifier>_<revision>.so`.

Parameters

out	*identifier*	The identifier provided to you by the project sponsor.
out	*revision*	A unique revision number for this submission. No two submission revision numbers may be the same, and subsequent submissions should only ever increase this value.
out	*email*	Point of contact email address.

Note

This method must return immediately.

3.9.3.3 getImplementation()

```
static std::shared_ptr<Interface> N2N::Interface::getImplementation ( )  [static]
```

Obtain a managed pointer to an implementation of this interface.

Returns

A managed pointer to the Interface subclass implementation.

3.9.3.4 identifyTemplateStageOne()

```
virtual ReturnStatus N2N::Interface::identifyTemplateStageOne (
                const std::string & searchID,
                const BiometricEvaluation::Memory::uint8Array & searchTemplate,
                const std::string & stageOneDataDirectory )  [pure virtual]
```

Search a template against the partial enrollment set.

Parameters

in	*searchID*	The ID of the search template. This ID does not identify subject, but is merely an identifier used to distinguish different searches performed by the system. It will be used as the input to identifyTemplateStageTwo().
in	*searchTemplate*	A template from makeSearchTemplate().
in	*stageOneDataDirectory*	This directory will have read-write access. The output information from identifyTemplateStageOne() that is needed in identifyTemplateStageTwo() is written in this directory. This directory will be unique for each search performed.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

All calls to combined identification functions (identifyTemplateStageOne() + identifyTemplate↩ StageTwo()) for a single `searchID` must return within 60 seconds for InputType::Capture and 300 seconds for InputType::Latent. If identifyTemplateStageOne() takes 55 seconds for searchID XYZ (InputType::Capture), identifyTemplateStageTwo() must complete within 5 seconds for the same search ID. `stageOneDataDir` will reside on a RAM disk to reduce the effects of I/O operations on this time requirement.

Attention

Multithreading and other multiprocessing techniques are absolutely not permitted. The testing application will be calling this method from multiple processes on the same node.
Unique filenames are required for all data written to `stageOneDataDirectory` for each `search↩ ID`. This can be accomplished by appending `nodeNumber` (from initIdentificationStageOne()) to the filename of any file written within `stageOneDataDirectory`.

3.9.3.5 identifyTemplateStageTwo()

```
virtual ReturnStatus N2N::Interface::identifyTemplateStageTwo (
              const std::string & searchID,
              const std::string & stageOneDataDirectory,
              std::vector< Candidate > & candidates )  [pure virtual]
```

Produce a candidate list from the results of all calls to identifyTemplateStageOne() for a particular search ID.

identifyTemplateStageOne() with searchID was called >= 1 times on separate nodes, ideally searching different subsets of the full enrolled set. In this method, the implementation should parse the results of the first search stage to form a final candidate list. This method will only be called once per searchID and only on a single node.

Parameters

`in`	*searchID*	The ID of the search template. This ID does not identify subject, but is merely an identifier used to distinguish different searches performed by the system.
`in`	*stageOneDataDirectory*	A read-only version of the data generated by identifyTemplateStageOne().
`out`	*candidates*	The candidate list.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

All calls to combined identification functions (identifyTemplateStageOne() + identifyTemplate↩StageTwo()) for a single `searchID` must return within 60 seconds for InputType::Capture and 300 seconds for InputType::Latent. If identifyTemplateStageOne() takes 55 seconds for searchID XYZ (InputType::Capture), identifyTemplateStageTwo() must complete within 5 seconds for the same search ID.
`candidates` will have `reserve()` called prior to calling this method.
There shall be [0,100] objects in `candidates` after the successful return of this method.
`candidates` shall be sorted by descending similarity score before returning.

Attention

Multithreading and other multiprocessing techniques are absolutely not permitted. The testing application will be calling this method from multiple processes on the same node.

3.9.3.6 initIdentificationStageOne()

```
virtual ReturnStatus N2N::Interface::initIdentificationStageOne (
                const std::string & configurationDirectory,
                const std::string & enrollmentDirectory,
                const InputType & inputType,
                const uint8_t nodeNumber )  [pure virtual]
```

Prepare for calls to identifyTemplateStageOne().

The function will be called to initialize each node that will contain a portion of the enrolled templates. The number of nodes will be the same as provided in finalizeEnrollment().

Parameters

`in`	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.
`in`	*enrollmentDirectory*	The top-level read-only directory in which all finalized enrollment data resides. The contents of this directory is identical to the enrollmentDirectory parameter from finalizeEnrollment(), but the path may not be the same.
`in`	*inputType*	The types of images that will be provided during all subsequent calls to identifyTemplateStageOne().
`in`	*nodeNumber*	Node number from nodes in finalizeEnrollment() that is being initialized. This parameter lets the callee know which piece of the enrolled templates to load into memory. Nodes are numbered 0 to (N - 1).

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method must complete with 5 minutes. Reasonable multithreading is permitted.

3.9.3.7 initIdentificationStageTwo()

```
virtual ReturnStatus N2N::Interface::initIdentificationStageTwo (
                const std::string & configurationDirectory,
                const std::string & enrollmentDirectory,
                const InputType & inputType )  [pure virtual]
```

Prepare for calls to identifyTemplateStageTwo().

This second stage of identification uses the results from identifyTemplateStageOne() to produce a candidate list for the search subject.

Parameters

in	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.
in	*enrollmentDirectory*	The top-level directory in which all finalized enrolled data resides. The directory will have read-only access.
in	*inputType*	The types of images that will be provided during all subsequent calls to identifyTemplateStageTwo().

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method must complete with 5 minutes. Reasonable multithreading is permitted.

3.9.3.8 initMakeEnrollmentTemplate()

```
virtual ReturnStatus N2N::Interface::initMakeEnrollmentTemplate (
              const std::string & configurationDirectory )  [pure virtual]
```

Prepare for calls to makeEnrollmentTemplate().

The function is called once by the testing application before N >= 1 calls to makeEnrollmentTemplate() on the current node. The implementation must tolerate execution of this initialization function and other N >= 1 calls to makeEnrollmentTemplate() running simultaneously and independently on the same and/or multiple machines.

Parameters

in	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method must complete with 5 minutes. Reasonable multithreading is permitted.

3.9.3.9 initMakeSearchTemplate()

```
virtual ReturnStatus N2N::Interface::initMakeSearchTemplate (
              const std::string & configurationDirectory,
              const InputType & inputType )  [pure virtual]
```

Prepare for calls to makeSearchTemplate().

The function is called once by the testing application before N >= 1 calls to makeSearchTemplate() on the current node. The implementation must tolerate execution of this initialization function and other N >= 1 calls to makeSearchTemplate() running simultaneously and independently on the same and/or multiple machines.

Parameters

in	*configurationDirectory*	A read-only directory containing vendor-supplied configuration parameters or run-time data files.
in	*inputType*	The types of images that will be provided during all subsequent calls to makeSearchTemplate().

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

90% of calls to this method must return in three seconds or less.
This method must complete with 5 minutes. Reasonable multithreading is permitted.

3.9.3.10 makeEnrollmentTemplate()

```
virtual ReturnStatus N2N::Interface::makeEnrollmentTemplate (
              const std::vector< FingerImage > & standardImages,
              const std::vector< BiometricEvaluation::Memory::uint8Array > & proprietaryImages,
              BiometricEvaluation::Memory::uint8Array & enrollmentTemplate )  [pure virtual]
```

Create an enrollment template for one subject.

This method provides one or more fingerprints from a subject and tasks the implementation with creating and returning an object that can represent this subject in an enrollment set.

Parameters

`in`	*standardImages*	One or more finger images from a single subject.
`in`	*proprietaryImages*	One or more proprietary representations of fingers, as returned from the participant's sensor.
`out`	*enrollmentTemplate*	A non-regulated representation of fingers for an enrollment set.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method should call BiometricEvaluation::Memory::uint8Array::resize() *before* any writes to `enrollmentTemplate` to ensure it is large enough to contain the write. This method should also call BiometricEvaluation::Memory::uint8Array::resize() *before* returning so that `enrollmentTemplate` is the exact required size. All BiometricEvaluation::Memory::uint8Array::size() bytes of `enrollment↩Template` will be provided to the N2N::Interface implementation during finalizeEnrollment().

90% of calls to this method must return in three seconds or less.

Attention

Multithreading and other multiprocessing techniques are absolutely not permitted. The testing application will be calling this method from multiple processes on the same node.

3.9.3.11 makeSearchTemplate()

```
virtual ReturnStatus N2N::Interface::makeSearchTemplate (
              const std::vector< FingerImage > & standardImages,
              const std::vector< BiometricEvaluation::Memory::uint8Array > & proprietaryImages,
              BiometricEvaluation::Memory::uint8Array & searchTemplate )  [pure virtual]
```

Create a search template for one subject.

This method provides one or more fingerprints from a subject and tasks the implementation with creating and returning an object that can represent this subject as a search initiator.

Parameters

`in`	*standardImages*	One or more finger images from a single subject.
`in`	*proprietaryImages*	One or more proprietary representations of fingers, as returned from the participant's sensor.
`out`	*searchTemplate*	A non-regulated representation of fingers used to initiate a search.

Returns

Completion status of the operation.

Exceptions

BiometricEvaluation::Error::Exception	There was an error processing this request, and the exception string may contain additional information.

Note

This method should call BiometricEvaluation::Memory::uint8Array::resize() *before* any writes to `searchTemplate` to ensure it is large enough to contain the write. This method should also call BiometricEvaluation::Memory::uint8Array::resize() *before* returning so that `searchTemplate` is the exact required size. All BiometricEvaluation::Memory::uint8Array::size() bytes of `enrollment↩Template` will be provided to the N2N::Interface implementation during finalizeEnrollment().

Attention

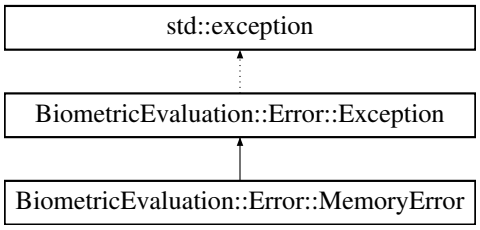
Multithreading and other multiprocessing techniques are absolutely not permitted. The testing application will be calling this method from multiple processes on the same node.

3.10 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 ()
  Construct a MemoryError object with the default information string.
- MemoryError (const std::string &info)
  Construct a MemoryError object with an information string appended to the default information string.

3.10.1 Detailed Description

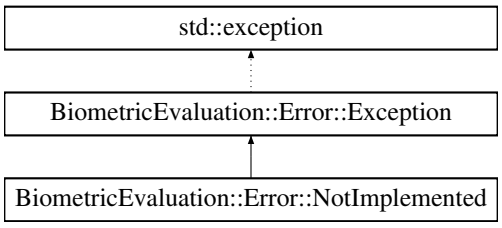
An error occurred when allocating an object.

3.11 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 ()
  Construct a NotImplemented object with the default information string.
- NotImplemented (const std::string &info)
  Construct a NotImplemented object with an information string appended to the default information string.

3.11.1 Detailed Description

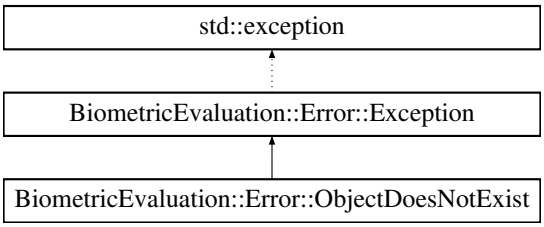
A NotImplemented object is thrown when the underlying implementation of this interface has not or could not be created.

3.12 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 ()

  Construct a ObjectDoesNotExist object with the default information string.
- ObjectDoesNotExist (const std::string &info)

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

3.12.1 Detailed Description

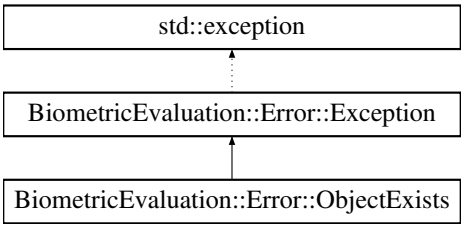
The named object does not exist.

3.13 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 ()
  Construct a ObjectExists object with the default information string.
- ObjectExists (const std::string &info)
  Construct a ObjectExists object with an information string appended to the default information string.

3.13.1 Detailed Description

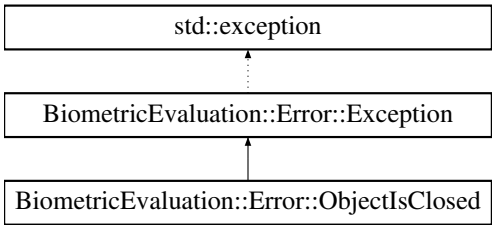
The named object exists and will not be replaced.

3.14 BiometricEvaluation::Error::ObjectIsClosed Class Reference

The object is closed.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectIsClosed:



Public Member Functions

- ObjectIsClosed ()
  Construct a ObjectIsClosed object with the default information string.
- ObjectIsClosed (const std::string &info)
  Construct a ObjectIsClosed object with an information string appended to the default information string.

3.14.1 Detailed Description

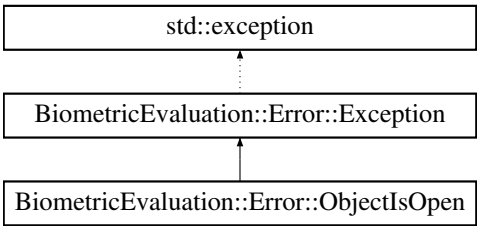
The object is closed.

3.15 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 ()
  Construct a ObjectIsOpen object with the default information string.
- ObjectIsOpen (const std::string &info)
  Construct a ObjectIsOpen object with an information string appended to the default information string.

3.15.1 Detailed Description

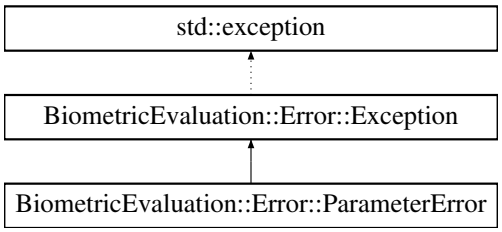
The object is already opened.

3.16 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 ()
  Construct a ParameterError object with the default information string.
- ParameterError (const std::string &info)
  Construct a ParameterError object with an information string appended to the default information string.

3.16.1 Detailed Description

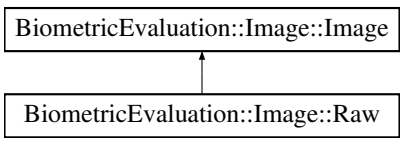
An invalid parameter was passed to a constructor or method.

3.17 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 uint32_t colorDepth, const uint16_t bitDepth, const Resolution resolution, const bool hasAlphaChannel)
- **Raw** (const BiometricEvaluation::Memory::uint8Array &data, const Size dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const Resolution resolution, const bool hasAlphaChannel)
- Memory::uint8Array getRawData () const

  Accessor for the raw image data.
- Memory::uint8Array getRawGrayscaleData (uint8_t depth) const

  Accessor for decompressed data in grayscale.

Additional Inherited Members

3.17.1 Detailed Description

An image with no encoding or compression.

3.17.2 Member Function Documentation

3.17.2.1 getRawData()

```
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

Error::DataError	Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

3.17.2.2 getRawGrayscaleData()

```
Memory::uint8Array BiometricEvaluation::Image::Raw::getRawGrayscaleData (
            uint8_t depth ) const  [virtual]
```

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 16, 8, or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError	Error decompressing image data.
Error::NotImplemented	Unsupported conversion based on source color depth.
Error::ParameterError	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.
Alpha channels are completely ignored when converting to grayscale.

Implements BiometricEvaluation::Image::Image.

3.18 BiometricEvaluation::IO::RecordStore::Record Struct Reference

Public Member Functions

- Record ()
  Default constructor.
- Record (const std::string &key, const Memory::uint8Array &data)
  Create a Record from the key and data.

Public Attributes

- std::string **key**
- Memory::uint8Array **data**

3.18.1 Constructor & Destructor Documentation

3.18.1.1 Record()

```
BiometricEvaluation::IO::RecordStore::Record::Record (
                const std::string & key,
                const Memory::uint8Array & data )
```

Create a Record from the key and data.

Parameters

in	*key*	The record's key.
in	*data*	The record's data (value).

3.19 BiometricEvaluation::IO::RecordStore Class Reference

A class to represent a data storage mechanism.

`#include <be_io_recordstore.h>`

Classes

- struct Record

Public Types

- enum Kind {
  Kind::BerkeleyDB, Kind::Archive, Kind::File, Kind::SQLite,
  Kind::Compressed, Kind::List, Kind::Default = BerkeleyDB }
  Possible types of RecordStore.
- using **Record** = struct Record
- using **iterator** = IO::RecordStoreIterator

Public Member Functions

- virtual std::string getDescription () const =0
  Obtain a textual description of the RecordStore.
- virtual unsigned int getCount () const =0
  Obtain the number of items in the RecordStore.
- virtual std::string getPathname () const =0
  Return the path name of the RecordStore.
- virtual void move (const std::string &pathname)=0
  Move the RecordStore.
- virtual void changeDescription (const std::string &description)=0
  Change the description of the RecordStore.
- virtual uint64_t getSpaceUsed () const =0
  Obtain real storage utilization.
- virtual void sync () const =0
  Synchronize the entire record store to persistent storage.
- virtual void insert (const std::string &key, const Memory::uint8Array &data)
  Insert a record into the store.
- virtual void insert (const std::string &key, const void *const data, const uint64_t size)=0
  Insert a record into the store.
- virtual void remove (const std::string &key)=0
  Remove a record from the store.
- virtual Memory::uint8Array read (const std::string &key) const =0
  Read a complete record from a store.

- virtual void replace (const std::string &key, const Memory::uint8Array &data)

  Replace a complete record in a RecordStore.
- virtual void replace (const std::string &key, const void *const data, const uint64_t size)

  Replace a complete record in a RecordStore.
- virtual uint64_t length (const std::string &key) const =0

  Return the length of a record.
- virtual void flush (const std::string &key) const =0

  Commit the record's data to storage.
- virtual RecordStore::Record sequence (int cursor=BE_RECSTORE_SEQ_NEXT)=0

  Sequence through a RecordStore, returning the key/data pairs.
- virtual std::string sequenceKey (int cursor=BE_RECSTORE_SEQ_NEXT)=0

  Sequence through a RecordStore, returning the key.
- virtual void setCursorAtKey (const std::string &key)=0

  Set the sequence cursor to an arbitrary position within the RecordStore, starting at key.
- 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, IO::Mode mode=Mode::ReadOnly)

  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 IO::RecordStore::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)

  Remove a RecordStore by deleting all persistant data associated with the store.
- static void mergeRecordStores (const std::string &mergePathname, const std::string &description, const IO::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

  The set of prohibited characters in a key: '/', '\', '', '@'.*
- static const int BE_RECSTORE_SEQ_START = 1

  Tell sequence() to sequence from beginning.
- static const int BE_RECSTORE_SEQ_NEXT = 2

  Tell sequence to sequence from current position.

3.19.1 Detailed Description

A class to represent a data storage mechanism.

A RecordStore is an abstraction that associates keys with a specific data item. 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.

3.19.2 Member Enumeration Documentation

3.19.2.1 Kind

```
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

3.19.3 Member Function Documentation

3.19.3.1 begin()

```
virtual iterator BiometricEvaluation::IO::RecordStore::begin ( )  [virtual], [noexcept]
```

Returns

Iterator to the first record.

3.19.3.2 changeDescription()

```
virtual void BiometricEvaluation::IO::RecordStore::changeDescription (
            const std::string & description )  [pure virtual]
```

Change the description of the RecordStore.

Parameters

in	*description*	The new description.

Exceptions

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

3.19.3.3 containsKey()

```
virtual bool BiometricEvaluation::IO::RecordStore::containsKey (
              const std::string & key ) const  [virtual]
```

Determines whether the RecordStore contains an element with the specified key.

Parameters

key	The key to locate.

Returns

True if the RecordStore contains an element with the key, false otherwise.

3.19.3.4 createRecordStore()

```
static std::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::createRecordStore (
              const std::string & pathname,
              const std::string & description,
              const IO::RecordStore::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	*pathname*	The directory of the store to be created.
in	*description*	The description of the store to be created.
in	*kind*	The kind of RecordStore to be created.

Returns

An managed pointer to the object representing the created store.

Exceptions

Error::ObjectDoesNotExist	The RecordStore does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

3.19.3.5 end()

```
virtual iterator BiometricEvaluation::IO::RecordStore::end ( )  [virtual], [noexcept]
```

Returns

Iterator past the last record.

3.19.3.6 flush()

```
virtual void BiometricEvaluation::IO::RecordStore::flush (
                const std::string & key ) const  [pure virtual]
```

Commit the record's data to storage.

Parameters

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

Exceptions

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

3.19.3.7 getCount()

```
virtual unsigned int BiometricEvaluation::IO::RecordStore::getCount ( ) const  [pure virtual]
```

Obtain the number of items in the RecordStore.

Returns

The number of items in the RecordStore.

3.19.3.8 getDescription()

```
virtual std::string BiometricEvaluation::IO::RecordStore::getDescription ( ) const  [pure virtual]
```

Obtain a textual description of the RecordStore.

Returns

The RecordStore's description.

3.19.3.9 getPathname()

```
virtual std::string BiometricEvaluation::IO::RecordStore::getPathname ( ) const  [pure virtual]
```

Return the path name of the RecordStore.

Returns

Where in the file system the RecordStore is located.

3.19.3.10 getSpaceUsed()

```
virtual uint64_t BiometricEvaluation::IO::RecordStore::getSpaceUsed ( ) const  [pure virtual]
```

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

3.19.3.11 insert() [1/2]

```
virtual void BiometricEvaluation::IO::RecordStore::insert (
              const std::string & key,
              const Memory::uint8Array & data )  [virtual]
```

Insert a record into the store.

Parameters

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

Exceptions

Error::ObjectExists	A record with the given key is already present.
Error::StrategyError	The RecordStore is opened read-only, or an error occurred when using the underlying storage system.

3.19.3.12 insert() [2/2]

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

Exceptions

Error::ObjectExists	A record with the given key is already present.
Error::StrategyError	The RecordStore is opened read-only, or an error occurred when using the underlying storage system.

3.19.3.13 length()

```
virtual uint64_t BiometricEvaluation::IO::RecordStore::length (
                const std::string & key ) const  [pure virtual]
```

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

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

3.19.3.14 mergeRecordStores()

```
static void BiometricEvaluation::IO::RecordStore::mergeRecordStores (
                const std::string & mergePathname,
```

```
const std::string & description,
const IO::RecordStore::Kind & kind,
const std::vector< std::string > & pathnames )  [static]
```

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

Parameters

in	*mergePathname*	The path name of the new RecordStore that will be created.
in	*description*	The text used to describe the new RecordStore.
in	*kind*	The kind of the new, merged RecordStore.
in	*pathnames*	Vector of path names to RecordStores to open. These are the RecordStores that will be merged to create the new RecordStore.

Exceptions

Error::ObjectExists	A RecordStore at mergePathname already exists.
Error::StrategyError	An error occurred when using the underlying storage system.

3.19.3.15 move()

```
virtual void BiometricEvaluation::IO::RecordStore::move (
                const std::string & pathname )  [pure virtual]
```

Move the RecordStore.

The RecordStore can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the RecordStore.

Exceptions

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

3.19.3.16 openRecordStore()

```
static std::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::openRecordStore (
                const std::string & pathname,
                IO::Mode mode = Mode::ReadOnly )  [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	*pathname*	The path name of the store to be opened.
in	*mode*	The type of access a client of this RecordStore has.

Returns

An object representing the existing store.

Exceptions

Error::ObjectDoesNotExist	The RecordStore does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

3.19.3.17 read()

```
virtual Memory::uint8Array BiometricEvaluation::IO::RecordStore::read (
              const std::string & key ) const  [pure 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.

Returns

The record associated with the key.

Exceptions

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

3.19.3.18 remove()

```
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

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

3.19.3.19 removeRecordStore()

```
static void BiometricEvaluation::IO::RecordStore::removeRecordStore (
                const std::string & pathname )  [static]
```

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

Parameters

in	*pathname*	The name of the existing RecordStore.

Exceptions

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

3.19.3.20 replace() [1/2]

```
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

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	The RecordStore is opened read-only, or an error occurred when using the underlying storage system.

3.19.3.21 replace() [2/2]

```
virtual void BiometricEvaluation::IO::RecordStore::replace (
                const std::string & key,
                const void *const data,
                const uint64_t size )  [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.
in	*size*	The size of the record, in bytes.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	The RecordStore is opened read-only, or an error occurred when using the underlying storage system.

3.19.3.22 sequence()

```
virtual RecordStore::Record BiometricEvaluation::IO::RecordStore::sequence (
                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 function 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

in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The record that is currently in sequence.

Exceptions

Error::ObjectDoesNotExist	End of sequencing.
Error::StrategyError	An error occurred when using the underlying storage system.

3.19.3.23 sequenceKey()

```
virtual std::string BiometricEvaluation::IO::RecordStore::sequenceKey (
              int cursor = BE_RECSTORE_SEQ_NEXT )  [pure virtual]
```

Sequence through a RecordStore, returning the key.

Sequencing means to start at some point in the store and return the key, then repeatedly calling the function to return the next key. 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

in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The key of the currently sequenced record.

Exceptions

Error::ObjectDoesNotExist	End of sequencing.
Error::StrategyError	An error occurred when using the underlying storage system.

3.19.3.24 setCursorAtKey()

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

Exceptions

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

3.19.3.25 sync()

```
virtual void BiometricEvaluation::IO::RecordStore::sync ( ) const  [pure virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

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

3.20 BiometricEvaluation::IO::RecordStoreIterator Class Reference

Generic ForwardIterator for all RecordStores.

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

Public Types

- using iterator_category = std::forward_iterator_tag
  Type of iterator.
- using value_type = RecordStore::Record
  Type when dereferencing iterators.
- using difference_type = std::ptrdiff_t
  Type used to measure distance between iterators.
- using pointer = value_type *
  Pointer to the type iterated over.
- using reference = value_type &
  Reference to the type iterated over.

Public Member Functions

- RecordStoreIterator ()=default
  Default constructor.
- RecordStoreIterator (IO::RecordStore *recordStore, bool atEnd)
  Constructor.
- RecordStoreIterator (const RecordStoreIterator &rhs)=default
  Default copy constructor.
- RecordStoreIterator (RecordStoreIterator &&rvalue)=default
  Default move constructor.
- ~RecordStoreIterator ()=default
  Default destructor.
- 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

  Default move assignment operator.

3.20.1 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().

3.20.2 Constructor & Destructor Documentation

3.20.2.1 RecordStoreIterator() [1/2]

```
BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator ( )  [default]
```

Default constructor.

Creates "end" iterator.

Note

Satisfies DefaultConstructible requirement.

3.20.2.2 RecordStoreIterator() [2/2]

```
BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator (
            IO::RecordStore * recordStore,
            bool atEnd )
```

Constructor.

Parameters

recordStore	Pointer to a RecordStore that will be iterated over.
atEnd	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.

3.20.3 Member Function Documentation

3.20.3.1 operator"!=()

```
bool BiometricEvaluation::IO::RecordStoreIterator::operator!= (
                const RecordStoreIterator & rhs )  [inline]
```

Non-equivalence operator.

Parameters

rhs	Reference to RecordStoreIterator being compared.

Returns

Whether or not this is not equivalent to rhs.

Note

Satisfies "i != j" is equivalent to "!(i == j)" condition of InputIterator.

3.20.3.2 operator*()

```
reference BiometricEvaluation::IO::RecordStoreIterator::operator* ( )
```

Returns

Reference to a Record.

3.20.3.3 operator+()

```
RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator+ (
                difference_type rhs )
```

Advance a variable number of arguments.

Parameters

rhs	Number of objects to advance (1 or more).

Returns

Self after advancing rhs objects.

3.20.3.4 operator++() [1/2]

```
RecordStoreIterator& BiometricEvaluation::IO::RecordStoreIterator::operator++ ( )
```

Returns

Self after advancing.

3.20.3.5 operator++() [2/2]

```
RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator++ (
            int postfix )
```

Returns

Copy of self before advancing.

3.20.3.6 operator+=()

```
RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator+= (
            difference_type rhs )
```

Advance a variable number of arguments.

Parameters

rhs	Number of objects to advance (1 or more).

Returns

Self after advancing rhs objects.

3.20.3.7 operator->()

```
pointer BiometricEvaluation::IO::RecordStoreIterator::operator-> ( )
```

Returns

A dereferenced Record.

3.20.3.8 operator==()

```
bool BiometricEvaluation::IO::RecordStoreIterator::operator== (
                const RecordStoreIterator & rhs )
```

Equivalence operator.

Parameters

rhs	Reference to RecordStoreIterator being compared.

Returns

Whether or not this is equivalent to rhs.

3.21 N2N::ReturnStatus Struct Reference

Information about the completion status of a method.

`#include <n2n.h>`

Public Member Functions

- ReturnStatus ()=default
  Constructor.
- ReturnStatus (const StatusCode code, const std::string info)
  Constructor.

Public Attributes

- StatusCode code {StatusCode::Success}
  Completion status of the returning method.
- std::string info {}
  Additional clarifying information about `code`.

3.21.1 Detailed Description

Information about the completion status of a method.

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

Note

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

3.21.2 Constructor & Destructor Documentation

3.21.2.1 ReturnStatus() [1/2]

```
N2N::ReturnStatus::ReturnStatus ( )  [default]
```

Constructor.

3.21.2.2 ReturnStatus() [2/2]

```
N2N::ReturnStatus::ReturnStatus (
            const StatusCode code,
            const std::string info )  [inline]
```

Constructor.

Parameters

in	*code*	The return status code.
in	*info*	The optional information string.

3.21.3 Member Data Documentation

3.21.3.1 code

```
StatusCode N2N::ReturnStatus::code {StatusCode::Success}
```

Completion status of the returning method.

3.21.3.2 info

```
std::string N2N::ReturnStatus::info {}
```

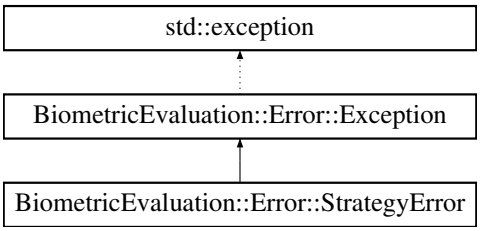
Additional clarifying information about `code`.

3.22 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 ()
  Construct a StrategyError object with the default information string.
- StrategyError (const std::string &info)
  Construct a StrategyError object with an information string appended to the default information string.

3.22.1 Detailed Description

A StrategyError object is thrown when the underlying implementation of this interface encounters an error.

Index