BIOMETRIC EVALUATION COMMON FRAMEWORK

PROGRAMMER'S GUIDE
VERSION 0.1

WAYNE SALAMON
GREGORY FIUMARA

IMAGE GROUP
INFORMATION ACCESS DIVISION
INFORMATION TECHNOLOGY LABORATORY

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

MARCH 13, 2015

Contents

Chapter 1

Introduction

This document describes the Biometric Evaluation Framework (BECommon) and application programming interfaces (API) used to support the evaluation of biometric software within the NIST Image Group [17].

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

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

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

This document describes each package and includes example code. The long form of this document includes reference sections containing auto-generated API documentation.

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

Chapter 2

Overview

The Biometric Evaluation Framework (BECommon) is a set of C++[22] classes, error codes, and design patterns used to create a common environment to provide logging, data management, error handling, and other functionality that is needed for many applications used in the testing of biometric software. The goals of the framework include:

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

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

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

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

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

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

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

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

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

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

Many packages in BECommon deal with biometric data record formats, including ANSI/NIST [3] records. In order to provide a general interface to several formats, BECommon represents the biometric data as derived from a source. For example, the `Finger` package contains classes that represent all information about a finger, including the source image and derived minutiae points. The `View` package combines the notions of a source image and derived information together into a single abstraction.

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

The `MPI` package provides wrappers around the Message Passing Interface (MPI) [15] libraries, handling all MPI communcation and error events. Many parallel applications can be greatly simplified, only implementing a few methods to process data.

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

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

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

Chapter 3

Framework

The `Framework` package is used to retrieve information about the Biometric Evaluation Framework itself, as well as to provide services through general purpose utility functions to other parts of the framework.

3.1 Versioning

Version numbers, the compiler used, and other framework metadata can be queried by applications. Versioning information is recorded in the BECommon `Makefile` and populated in the function implentation at compile-time.

Listing 3.1: Using the `Framework` API

```
/* "Framework Version: 0.4" */
std::cout << "Framework Version: " << BE::Framework::getMajorVersion() << "." <<
    BE::Framework::getMinorVersion() << std::endl;

/* "Compiler Used: clang v5.1.0" */
std::cout << "Compiler Used: " << BE::Framework::getCompiler() << " v" <<
    BE::Framework::getCompilerVersion() << std::endl;

/* "Date/Time Compiled: Jan 24 2014 12:16:01" */
std::cout << "Date/Time Compiled: " << BE::Framework::getCompileDate() << " " <<
    BE::Framework::getCompileTime() << std::endl;
```

3.2 Enumerations

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

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

Listing 3.2: `Framework::Enumeration`

```
/*
 * color.h
```

```
 */

enum class Color
{
        Black,
        Blue,
        Green
};

/*
 * color.cpp
 */

#include <be_framework_enumeration.h>

template<>
const std::map<Color, std::string>
BiometricEvaluation::Framework::EnumerationFunctions<Color>::enumToStringMap {
        {Color::Black, "Black"},
        {Color::Black, "Blue"},
        {Color::Green, "Green"}
};

/*
 * application.cpp
 */

#include <color.h>

/* "Black" */
std::cout << to_string(Color::Black) << std::endl;
/* "2" */
std::cout << to_int_type(Color::Green) << std::endl;
/* Color::Blue */
Color color = to_enum<Color>("Blue");
```

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

Chapter 4

Memory

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

4.1 AutoBuffer

The Biometric Evaluation Framework is designed to interoperate with existing C code that has its own memory management techniques, e.g. NIST Biometric Image Software [16]. In these cases, functions exist to allocate and free blocks of memory, and these calls must be made by the applications which use those libraries. To assist BECommon clients that use these existing libraries, the `AutoBuffer` class wraps the C memory management functions, guaranteeing the release of C objects when the AutoBuffer goes out of scope.

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

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

Listing 4.1: Using the `AutoBuffer`

```
#include <be_memory_autobuffer.h>
#include <iostream>
extern "C" {
  #include <an2k.h>
}

int
main(int argc, char* argv[]) {


    /*
     * alloc_ANSI_NIST(), free_ANSI_NIST(), and copy_ANSI_NIST()
     * are functions in the NBIS AN2K library.
     */
    Memory::AutoBuffer<ANSI_NIST> an2k =
        Memory::AutoBuffer<ANSI_NIST>(&alloc_ANSI_NIST,
            &free_ANSI_NIST, &copy_ANSI_NIST);
    if (read_ANSI_NIST(fp, an2k) != 0) {
            cerr << "Could not read AN2K file." << endl;
            return (EXIT_FAILURE);
```

```
21      }
22
23      for (int i = 1; i < an2k->num_records; i++) {
24          // process the ANSI/NIST record ...
25      }
26  }
```

4.2 AutoArray

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

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

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

Listing 4.2: Using `AutoArray` s with `RecordStore` s

```
1  #include <be_io_dbrecstore.h>
2  #include <be_memory_autoarray.h>
3
4  #include <iostream>
5
6  using namespace BiometricEvaluation;
7
8  int
9  main(
10     int argc,
11     char *argv[])
12 {
13         IO::DBRecordStore rs("db_recstore", ".", IO::READONLY);
14
15         uint64_t value_size = 0;
16         string key("");
17         Memory::AutoArray<uint8_t> value;
18         for (bool stop = false; stop == false; ) {
19                 try {
20                         // Non-destructively resize the AutoArray to hold
21                         // the next record.
22                         value.resize(rs.sequence(key, NULL));
23
24                         // Read the record into the AutoArray (treats the
25                         // AutoArray as a pointer).
26                         rs.read(key, value);
27
28                         // Do something with value.
29                         std::cout << "Key " << key << " has a value of " <<
30                             value.size() << " bytes" << std::endl;
```

```
31                  } catch (Error::ObjectDoesNotExist) {
32                          stop = true;
33                  }
34          }
35
36          return (0);
37  }
```

`AutoArray` is adapted from "`c_array` " [22, 496].

4.3 IndexedBuffer

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

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

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

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

Listing 4.3: Using the `IndexedBuffer`

```
1  #include <be_memory_autoarray.h>
2  #include <be_memory_indexedbuffer.h>
3
4  int
5  main(int argc, char* argv[]) {
6
7          uint64_t size = IO::Utility::getFileSize("BiometricRecord");
8          FILE *fp = std::fopen("BiometricRecord", "rb");
9          Memory::IndexedBuffer iBuf(size);
10         fread(iBuf, 1, size, fp);
11         fclose(fp);
12         Memory::IndexedBuffer iBuf(recordData, recordData.size());
13
14         uint32_t lval;
15         uint16_t sval;
16
17         /*
18          * Record is big-endian:
19          * ---------------------------------
20          * | NAME | LENGTH | ID |    ...    |
21          * ---------------------------------
22          *      4       4       2
23          */
24
25         /* Read a 4-byte C string */
26         lval = iBuf.scanU32Val();          /* Format ID */
27         char *cptr = (char *)&lval;
```

```
28          string s(cptr);
29
30          /* Read a 4-byte length */
31          lval = iBuf.scanBeU32Val()
32
33          /* Read a 2-byte ID */
34          sval = iBuf.scanBeU16Val();
35  }
```

Chapter 5

Error Handling

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

5.1 Biometric Evaluation Exceptions

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

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

Listing 6.2 on page 17 shows an example of exception handling when using the logging classes described in Section 6.3 on page 16.

5.2 Signal Handling

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

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

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

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

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

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

Listing 5.1: Using the `SignalManger`

```
#include <be_error_signal_manager.h>
using namespace BiometricEvaluation;

int main(int argc, char *argv[])
{
        Error::SignalManager *sigmgr = new Error::SignalManager();

        BEGIN_SIGNAL_BLOCK(sigmgr, sigblock1);
        // code that may result in signal generation
        END_SIGNAL_BLOCK(asigmgr, sigblock1);
        if (sigmgr->sigHandled()) {
                // log the event, etc.
        }
}
```

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

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

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

Listing 5.2: Specifying Signals to the `SignalManger`

```
#include <be_error_signal_manager.h>
using namespace BiometricEvaluation;

int main(int argc, char *argv[])
{
    Error::SignalManager *sigmgr = new Error::SignalManager();

    sigset_t sigset;
    sigemptyset(&sigset);
    sigaddset(&sigset, SIGUSR1);
    sigmgr->setSignalSet(sigset);

    FILE *fp = fopen( ... );
    BEGIN_SIGNAL_BLOCK(sigmgr, sigblock2);
        // code that may result in signal generation
        fclose(fp);
    END_SIGNAL_BLOCK(asigmgr, sigblock2);
```

```
18      if (sigmgr->sigHandled()) {
19          cout << "SIGUSR1 occurred." << endl;
20          fclose(fp);
21      }
22  }
```

Chapter 6

Input/Output

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

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

6.1 Utility

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

6.2 Record Management

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

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

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

Use of the `RecordStore` abstraction allows applications to switch storage strategy by changing a few lines of code. Furthermore, error handling is consistent for all strategies by the use of common exceptions.

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

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

Listing 6.1: Using a `RecordStore`

```
#include <be_io_dbrecstore.h>
int
main(int argc, char* argv[]) {

    IO::DBRecordStore *rs;
    try {
        rs = new IO::DBRecordStore("myRecords", "My Record Store");
    } catch (Error::Exception& e) {
        cout << "Caught " << e.what() << endl;
        return (EXIT_FAILURE);
    }
    std::unique_ptr<IO::DBRecordStore> urs(rs);

    try {
        Memory::uint8Array theData;

        theData = getSomeData();
        urs->insert("key1", theData);

        theData = getSomeData();
        urs->insert("key2", theData);

    } catch (Error::Exception& e) {
        cout << "Caught " << e.what() << endl;
        return (EXIT_FAILURE);
    }

    // Some more processing where new data for a key comes in ...
    theData = getSomeData();
    urs->replace("key1", theData);

    // Obtain the data for all keys ...
    string theKey;
    while (true) {
        uint64_t len = rs->sequence(theKey, theData);
        cout << "Read data for key " << theKey << " of length " << len << endl;
    }
    // The data for the key is no longer needed ...
    urs->remove("key1");
}
```

6.3 Logging

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

The logging classes in the `IO` package provide a straight-forward method for applications to record their progress without the need to manage the low-level storage details. Management of the log messages to the backing store is done within the `Logsheet` implementations. `Logsheet` specifies the common interface to

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

A `Logsheet` is an output stream (subclass of `std::ostringstream`), and therefore can handle built-in types and any class that supports streaming. Each entry is numbered by the `Logsheet` class when written to the log. A call to the `newEntry()` method commits the current entry to the log, and resets the write position to the beginning of the entry buffer.

In addition to streaming by using the `Logsheet::<<` operator, applications can directly commit an entry to the log file by calling the `write()` method, thereby not disrupting the entry that is being formed. After an entry is committed, the entry number is automatically incremented. `Logsheet` also supports the writing of "debug" and comment entries. Each entry is prefixed with a letter code indicating the type.

6.3.1 FileLogsheet

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

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

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

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

```
#include <be_io_filelogcabinet.h>
using namespace BiometricEvaluation;
using namespace BiometricEvaluation::IO;

FileLogCabinet *lc;
try {
    lc = new FileLogCabinet(lcname, "A Log Cabinet", "");
} catch (Error::ObjectExists &e) {
    cout << "The Log Cabinet already exists." << endl;
    return (-1);
} catch (Error::StrategyError& e) {
    cout << "Caught " << e.what() << endl;
    return (-1);
}
std::unique_ptr<FileLogCabinet> ulc(lc);
try {
    ls = alc->newLogsheet("log01", "Log Sheet in Cabinet");
} catch (Error::ObjectExists &e) {
    cout << "The log sheet already exists." << endl;
    return (-1);
} catch (Error::StrategyError& e) {
    cout << "Caught " << e.what() << endl;
    return (-1);
}
ls->setAutoSync(true);  // Force write of every entry when finished
int i = ...
*ls << "Adding an integer value " << i << " to the log." << endl;
ls->newEntry();         // Forces the write of the current entry
.........
delete ls;
return;                 // The LogCabinet is destructed by the unique_ptr
```

6.3.2 SysLogsheet

The `SysLogsheet` is an implementation of `Logsheet` which writes log entries to a system logger service. Objects of this class are created with a URL starting with `syslog://`. When using a system logger, the URL must give the hostname of the logger as well as the network port: `syslog://node00:4315` for example. The system logger must understand the Syslog protocol as specified in RFC5424 [23].

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

6.4 Properties

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

Listing 6.3: Using a `Properties` Object

```
IO::Properties *props;
string fname = "test.prop";
try {
    props = new IO::Properties(fname);
} catch (Error::StrategyError &e) {
    cerr << "Caught " << e.what()  << endl;
    return;
} catch (Error::FileError& e) {
    cerr << "A file error occurred: " << e.what() << endl;
    return;
}
props->setProperty("foo", "bar");
props->setProperty("theAnswer", "42");
    :
    :
    :
try {
    int64_t theAnswer = props->getProperty("theAnswer");
    cout << "The answer is " << theAnswer << endl;
} catch (Error::ObjectDoesNotExist &e) {
    cerr << "The answer is elusive." << endl;
    return;
}
string fooProp = props->getProperty("foo");
cout << "Foo is set to " << fooProp << endl;
    :
    :
    :
try {
    props->removeProperty("foo");
} catch (Error::ObjectDoesNotExist &e) {
    cerr << "Failed to remove property." << endl;
}
```

6.5 Compressor

Support for data compression and decompression can be found in the Biometric Evaluation Framework through the `Compressor` class hierarchy. `Compressor` is an abstract base class defining several pure-virtual methods for compression and decompression of buffers and files. Derived classes implement these methods and can be instantiated through the factory method in the base class. As such, children should also be enumerated within `Compressor::Kind`. The Biometric Evaluation Framework comes with an example, `GZIP`, which compresses and decompresses the `gzip` format through interaction with `zlib` [4].

Listing 6.4: Using a `Compressor` Object

```
shared_ptr<IO::Compressor> compressor;
Memory::uint8Array compressedBuffer, largeBuffer = /* ... */;
try {
        compressor = IO::Compressor::createCompressor(Compressor::Kind::GZIP);
        /* Overloaded for all combination of buffer and file */
        compressor->compress("largeInputFile", "compressedOutputFile");
        compressor->compress(largeBuffer, compressedBuffer);
} catch (Error::Exception &e) {
        cerr << "Could not compress (" << e.what() << ')' << endl;
}
```

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

Listing 6.5: Setting `Compressor` Options

```
shared_ptr<IO::Compressor> compressor =
    IO::Compressor::createCompressor(Compressor::Kind::GZIP);

/* A large GZIP chunk size can speed operations on systems with copious RAM */
compressor->setOption(IO::GZIP::CHUNK_SIZE, 32768);
```

Chapter 7

Time and Timing

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

7.1 Elapsed Time

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

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

Listing 7.1: Using the `Timer`

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

7.2 Limiting Execution Time

The `Watchdog` class allows applications to control the amount of time that a block of code has to execute. The time can be *real* (i.e. "wall") time, or *process* time (not available on Windows). One typical usage for a `Watchdog` timer is when a call is made to a function that may never return, due to problems processing an input biometric image.

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

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

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

Listing 7.2: Using the `Watchdog`

```
#include <be_time_watchdog.h>
int main(int argc, char *argv[])

    Time::Watchdog theDog = new Time::Watchdog(Time::Watchdog::PROCESSTIME);
    theDog->setInterval(300);    // 300 microseconds

    Time::Timer timer;

    BEGIN_WATCHDOG_BLOCK(theDog, watchdogblock1);
        timer.start();
        // Do something that may take more than 300 usecs
        timer.stop();
        cout << "Total time was " <<  timer.elapsed() << endl;
    END_WATCHDOG_BLOCK(theDog, watchdogblock1);
    if (theDog->expired()) {
        timer.stop();
        cerr << "That took too long." << endl;
    }
{
}
```

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

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

Chapter 8

Process Information and Control

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

8.1 Process Statistics

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

Listing 8.1: Gathering Process Statistics

```
1  #include <be_error_exception.h>
2  #include <be_process_statistics.h>
3  using namespace BiometricEvaluation;
4
5  int main(int argc, char *argv[])
6  {
7      Process::Statistics stats;
8      uint64_t userstart, userend;
9      uint64_t systemstart, systemend;
10     uint64_t diff;
11     try {
12         stats.getCPUTimes(&userstart, &systemstart);
13
14         // Do some long processing....
15
16         stats.getCPUTimes(&userend, &systemend);
17         diff = userend - userstart;
18         cout << "User time elapsed is " << diff << endl;
19         diff = systemend - systemstart;
20         cout << "System time elapsed is " << diff << endl;
21     } catch (Error::Exception) {
22         cout << "Caught " << e.getInfo() << endl;
23     }
24
25 }
```

In addition to using the `Process` API to gather statistics to be returned from the function call, the API provides a means to have a "standard" set of statistics logged either synchronously or asynchronously to a `LogSheet` (See Section 6.3 on page 16) contained within a `LogCabinet`. Applications can start and stop logging at will to this `LogSheet`. Post-mortem analysis can then be done on the entries in the log. Listing 8.2 shows the use of logging.

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

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

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

Listing 8.2: Logging Process Statistics

```
#include <be_error_exception.h>
#include <be_io_logcabinet.h>
#include <be_process_statistics.h>
using namespace BiometricEvaluation;

int main(int argc, char *argv[])
{
    IO::LogCabinet lc("statLogCabinet", "Cabinet for Statistics", "");

    Process::Statistics *logstats;
    try {
        logstats = new Process::Statistics(&lc);
    } catch (Error::Exception &e) {
        cout << "Caught " << e.getInfo() << endl;
        return (EXIT_FAILURE);
    }
    try {
        while (some_processing_to_do) {
            // Do the work
            // Synchronously log after the work is done.
            logstats->logStats();
        }
    } catch (Error::Exception &e) {
        cout << "Caught " << e.getInfo() << endl;
        delete logstats;
        return (EXIT_FAILURE);
    }

    // Set up asynchronous logging, every second
    try {
        logstats->startAutoLogging(1);
    } catch (Error::ObjectExists &e) {
        cout << "Caught " << e.getInfo() << endl;
        delete logstats;
        return (EXIT_FAILURE);
    }

    // Do some other work
```

```
39
40      // Stop logging
41      logstats->stopAutoLogging();
42      delete logstats;
43  }
```

8.2 Process Management

During a biometric evaluation or other long-running CPU-bound task, it's beneficial to make efficient use of all the hardware available on the system. Applications can take advantage of a multi-core machine, for example. BECommon aims to simply this by abstracting the usage of process and thread creation to run multiple instances of the same function in parallel.

8.2.1 Manager

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

8.2.2 Worker

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

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

Listing 8.3: A Responsible `Worker` Implementation

```
1  #include <cstdlib>
2  #include <tr1/memory>
3  #include <queue>
4
5  #include <restaurant.h>
6
7  #include <be_process_forkmanager.h>
8
9  using namespace std;
10 using namespace BiometricEvaluation;
```

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

After a manager starts its workers, the manager has the option of waiting until all `Worker` s exit `workerMain()` before continuing code execution. If not waiting, there are several methods the manager can perform to keep track of the status of the workers. Even if not waiting for workers to return, a responsible manager will wait a reasonable amount of time for workers to `return` before application termination. An example of this reasonable waiting period can be seen in Listing 8.4 on the next page.

8.2.3 WorkerController

The final piece of the process management puzzle is the `WorkerController` hierarchy. This class decorates and mediates communication between the `Manager` and the `Worker`. `WorkerController` objects may only be instantiated by a `Manager` object. All communications to the `Worker` (e.g. `isWorking()`) should be delegated through the `WorkerController`. If defining a new `Manager`, note that the `Worker`

`Controller` may seem unnecessary for the parallelization technique being employed. It's true that some parallelization techniques may not require this "middle-man" approach, but others do. Do not be concerned if a `WorkerController` implementation ends up being nothing more than a "pass-thru" to the `Worker`.

Listing 8.4 is a continuation of Listing 8.3 on page 25 demonstraiting the use of `Manager` s and `Worker Controller` s.

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

```
int
main(
    int argc,
    char *argv[])
{
        static const uint32_t numEmployees = 3;
        int status = EXIT_FAILURE;

        tr1::shared_ptr<Process::Manager> shiftLeader(new Process::ForkManager);
        queue<Customer*> *customers = new queue<Customer*>();

        /* Create Employees (Workers/WorkerControllers) */
        tr1::shared_ptr<Process::WorkerController> employees[numEmployees];
        for (uint32_t i = 0; i < numEmployees; i++) {
                employees[i] = shiftLeader->addWorker(
                    tr1::shared_ptr<ResponsibleEmployeeTask>(
                    new ResponsibleEmployeeTask()));

                /* Assign employees to each Task */
                employees[i]->setParameter("employee",
                    tr1::shared_ptr<Employee>(new Employee()));
                employees[i]->setParameter("customers",
                    tr1::shared_ptr< queue<Customer*> >(customers);
        }

        /* Employees start serving customers while shift leader manages */
        shiftLeader->startWorkers(false);

        /* Customers enter the queue... */
        queue<Restaurant::AdministrativeTasks> adminTasks;
        adminTasks.push("Inventory");
        adminTasks.push("Customer Complaints");
        adminTasks.push("Clean Dining Room");

        while (shiftLeader->getNumActiveWorkers() != 0) {
                shiftLeader->doTask(adminTasks.front());
                adminTasks.pop();
        }

        /* ...end of the day */
        for (uint32_t i = 0; i < numEmployees; i++)
                if (employees[i]->isWorking())
                        shiftLeader->stopWorker(employees[i]);

        /*
         * Wait a reasonable amount of time before locking up for the night
         * (in this case, indefinitely).
         */
```

```
49          while (shiftLeader->getNumActiveWorkers() > 0)
50                  sleep(1);
51
52          shiftLeader->armAlarmAndExit();
53
54          status = EXIT_SUCCESS;
55          return (status);
56  }
```

8.2.4 Communications

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

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

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

Listing 8.5: `Worker` Communication

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

Listing 8.6: `Manager` Communication

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

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

Chapter 9

System

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

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

Listing 9.1: Using the `System` CPU Count Information

```
#include <iostream>
#include <be_system.h>

using namespace BiometricEvaluation;

int
main(int argc, char* argv[]) {

    // perform some application setup ...

    uint32_t cpuCount;
    uint64_t memSize, vmSize;
    try {
        cpuCount =  System::getCPUCount();
        cpuCount--;    // subtract one CPU for the parent process
        memSize = System::getRealMemorySize();
        Process::Statistics::getMemorySizes(NULL, &vmSize, NULL, NULL, NULL);
        memSize -= vmSize;   // subtract off memory used by parent

        // Give each child a fraction of the memory
        spawnChildren(cpuCount, memSize / cpuCount);
    } catch (Error::NotImplemented) {
            cout << "Running a single process only." << endl;
    }

    // processing done by parent ...
}
```

Chapter 10

Image

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

10.1 The Image Namespace

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

The same applies to images encapsulated in biometric records. Although some biometic records have fields for image attributes like dimensions and resolution, the corresponding fields of an `Image` class are **not** populated with their contents. The `Image` namespace data types *are* used outside of the namespace, such as in finger views, to retrieve image attributes stored as part of the biometric record. Applications can compare those values against the values within the `Image` object, as in most cases those values are taken directly from the underlying image data. See Chapter 15 on page 49 for more information on image-based biometric records.

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

10.2 The Image Class

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

- Original binary data
- Compression algorithm
- Decompressed ("raw") format binary data (grayscale, full color)
- Depth

- Dimensions (width, height)
- Resolution (horizontal, vertical)

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

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

10.3 Raw Image

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

10.4 JPEG

The `JPEG` class represents an image encoded according to the JPEG image standard [11]. Decompression and grayscale conversion are accomplished via `libjpeg` [9].

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

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

10.5 JPEGL

Similar to `JPEG`, the `JPEGL` class performs `Image` class services for lossless JPEG encoded images. JPEGL decompression is performed by NIST Biometric Image Software 's `libjpegl` [16].

10.6 JPEG2000

The `JPEG2000` class provides Image class functionality to JPEG 2000-encoded images [10]. The class makes an attempt to support the following JPEG 2000 codecs:

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

Decompression is provided by the OpenJPEG library (`libopenjpeg`) [14]. `JPEG2000` also provides a static function to test whether or not an image appears to be JPEG 2000-encoded.

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

Errors within `libopenjpeg` will be caught and rethrown as `Exception` s.

10.7 NetPBM

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

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

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

10.8 PNG

The `PNG` class represents an image encoded according to the PNG image standard [6]. Decompression is provided by `libpng` [20].

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

10.9 WSQ

Images encoded in the WSQ-image standard [24] are represented by the `WSQ` class. The WSQ decompressor found in NIST Biometric Image Software [16], `libwsq`, is used by this class. The class provides a static function to determine whether or not an image appears to be encoded in the WSQ format.

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

Chapter 11

Video

The `Video` package is used to access video (and, in the future, audio) streams from containers in several formats, such as MPEG4. The classes in this package rely on the FFmpeg [7] libraries to de-multiplex video streams from a container, and to decode the streams and retrieve the frames from the video.

11.1 Container

`Container` objects can be instantiated in three ways:

1. With a filename: Memory usage will equal to the size of the container stream;
2. With a `AutoArray::uint8Array`: Memory usage will be twice that of the size of the container stream;
3. With a `std::shared_ptr` wrapping a `AutoArray::uint8Array`: Memory usage can equal to the size of the container stream. Applications must not modify the container data.

By careful coding, the application can prevent duplicate copies of the container buffer when using method three. By taking advantage of C++ 2011 move semantics, BECommon and the application avoid duplicate copies. See Listing 11.1 for examples of using all three methods.

11.2 Stream

`Stream` objects represent a single video stream within the container and provide access to individual frames from the video stream. In addition, these frames can be retrieved at their native size, or can be scaled to a different size. Frames can be returned as 24-bit red/green/blue images, grayscale, or two-color monochrome.

`Stream` objects can be obtained only from a `Container` object. The reason for this is that video frames must be pulled from a stream that is de-multiplexed from the container stream shared with the `Container` object. Future versions of BECommon may allow for `Stream` s to be directly instantiated with coded video streams.

Listing 11.1 shows the use of `Container` and `Stream`.

Listing 11.1: Using the Video Framework

```
1 #include <iostream>
2 #include <be_memory_autoarray.h>
3 #include <be_io_utility.h
4 #include <be_video_container.h>
5
```

```
6  using namespace BiometricEvaluation;
7  using namespace std;
8
9  int
10 main(int argc, char* argv[])
11 {
12         std::unique_ptr<Video::Container> pvc;
13
14         std::string filename = "./test_data/2video1audio.mp4";
15         if ((argc != 1) && (argc != 2)) {
16                 cerr << "usage: " << argv[0] << " [filename]" << endl
17                     << "If <filename> is not given, " << filename
18                     << " is used instead." << endl;
19                 return (EXIT_FAILURE);
20         }
21         if (argc == 2)
22                 filename = argv[1];
23
24         cout << "Construct an program stream from file "
25             << filename << endl;
26         /*
27          * Three ways to open the container:
28          * 1) Have the framework open the file directly;
29          * 2) Read the file into a local buffer and give that to the framework;
30          * 3) Read the file into a buffer wrapped in a shared pointer and pass
31          *    that to the framework.
32          */
33         try {
34 //              pvc.reset(new
35 //                  Video::Container(filename));
36
37 //              Memory::uint8Array buf =
38 //                  IO::Utility::readFile(filename);
39 //              pvc.reset(new Video::Container(buf));
40
41                 std::shared_ptr<Memory::uint8Array> buf;
42                 buf.reset(new Memory::uint8Array(
43                     IO::Utility::readFile(filename)));
44                 pvc.reset(new Video::Container(buf));
45         } catch (Error::Exception &e) {
46                 cout << "Caught: " << e.whatString() << endl;
47                 return (EXIT_FAILURE);
48         }
49
50         cout << "Video Count: " << pvc->getVideoCount() << endl;
51
52         std::unique_ptr<Video::Stream> stream;
53         /*
54          * Open the first video stream.
55          */
56         try {
57                 stream = pvc->getVideoStream(1);
58         } catch (Error::Exception &e) {
59                 cerr << "Could not retrieve video stream: " << e.whatString()
60                     << endl;
61                 return (EXIT_FAILURE);
```

```
62          }
63          /*
64           * Read all the frames, one at a time, scaled down and converted
65           * to 8-bit grayscale.
66           */
67          float scaleFactor = 0.5;
68          Image::PixelFormat pixelFormat = Image::PixelFormat::Gray8;
69          stream->setFrameScale(scaleFactor, scaleFactor);
70          stream->setFramePixelFormat(pixelFormat);
71          uint64_t expectedCount = stream->getFrameCount();
72
73          cout << "First video stream: " << stream->getFPS() << " FPS, "
74              << expectedCount << " frames." << endl;
75          /*
76           * The frame count can be zero, meaning unknown. If that is the case,
77           * loop until a parameter error is indicated.
78           */
79          if (expectedCount == 0)
80                  expectedCount = 99999999;
81          uint64_t count = 0;
82          for (uint64_t f = 1; f <= expectedCount; f++) {
83                  try {
84                          auto frame = stream->getFrame(f);
85                          count++;
86                          /* Do something with the frame */
87                  } catch (Error::ParameterError &e) {
88                          cout << "No more frames.";
89                          break;
90                  } catch (Error::Exception &e) {
91                          std::cout << "Caught " << e.whatString() << endl;
92                          return (EXIT_FAILURE);
93                  }
94          }
95          cout << "Retrieved " << count << " frames." << endl;
96          return (EXIT_SUCCESS);
97  }
```

Chapter 12

Text

The `Text` package consists of functions to perform common operations on `string` s and `char` arrays. Many of the operations may be considered "trivial," but are used often enough within the Biometric Evaluation Framework and other applications that a common implementation in BECommon is more than warranted. A complete listing of functions is available in the documentation appendix for `BiometricEvaluation::Text`2.

Listing 12.1 shows how to use the `split()` function from the `Text` package. `split()` can separate a `string` into tokens delimited by a character, useful for processing comma- or space-separated text files (such files could be produced by a `LogSheet` (Section 6.3 on page 16), for instance). Here, a text file containing metadata for an image is being parsed, perhaps to be passed to the `RawImage` constructor (Section 10.3 on page 34).

Listing 12.1: Tokenizing a `string`

```
/* Definition of input strings */
static const vector<string>::size_type filenameToken = 0;
static const vector<string>::size_type widthToken = 1;
static const vector<string>::size_type heightToken = 2;
static const vector<string>::size_type depthToken = 3;

/* Split the string, presumably input from a file */
string input = "/mnt/raw\\ images/1.raw 500 500 8";
vector<string> tokens = Text::split(input, ' ', true);

/* Assign the retrieved tokens */
string filename;
uint32_t width, height, depth;
try {
        filename = tokens.at(filenameToken);    /* "/mnt/raw images/1.raw" */
        width = atoi(tokens.at(widthToken).c_str());    /* "500" */
        height = atoi(tokens.at(heightToken).c_str());  /* "500" */
        depth = atoi(tokens.at(depthToken).c_str());    /* "8" */
} catch (out_of_range) {
        throw Error::FileError("Malformed input");
}
```

Notice the `true` parameter to `split()` in Listing 12.1. This instructs `split()` to not tokenize based on an escaped delimiter. If `false`, the first token would be split into two at the presence of the delimiter.

`Text` also contains functions to perform hashing via `OpenSSL`. A two-line program that emulates the command-line `md5sum` program is shown in Listing 12.2. Changing the digest parameter to `"sha1"` would make the program emulate `` `openssl sha1` ``.

Listing 12.2: `md5sum` via BECommon

```
#include <cstdlib>
#include <iostream>

#include <be_io_utility.h>
#include <be_text.h>
#include <be_memory_autoarray.h>

using namespace std;
using namespace BiometricEvaluation;

int
main(
    int argc,
    char *argv[])
{
        if (argc == 0)
                return (EXIT_FAILURE);

        try {
                Memory::uint8Array file = IO::Utility::readFile(argv[1]);
                cout << Text::digest(file, file.size(), "md5") << "  " <<
                    argv[1] << endl;
        } catch (Error::Exception) {
                return (EXIT_FAILURE);
        }

        return (EXIT_SUCCESS);
}
```

Chapter 13

Feature

The `Feature` package contains those items that relate to the representation of biometric features, such as fingerprint minutiae, facial features (eyes, etc.), and related information. Objects of these class types are typically associated with `View` (Chapter 15 on page 49) or `DataInterchange` (Chapter 16 on page 51) objects. For example, a minutiae object is usually obtained from a finger view, which may have been obtained from a data interchange object representing an entire biometric record for an individual.

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

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

13.1 ANSI/NIST Features

The ANSI/NIST [3] standard defines several features represented as data elements within a record. Fingerprint and palm minutiae is contained within Type-9 record. The `AN2K7Minutiae` class, contained in the Feature package, represents a single Type-9 record. An object of this class can be constructed directly from a complete ANSI/NIST record. However, it is more common for an application to retrieve these objects from the `AN2KView` object defined in the `Finger` package (Chapter 14 on page 45).

See Listing 14.1 on page 46 for a complete example of how to obtain the fingerprint minutiae data from an ANSI/NIST record.

13.2 ISO/INCITS Features

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

Listing 14.2 on page 47 shows how to create a view object for the fingerprint minutiae record contained in a file.

Chapter 14

Finger

One of the most commonly used biometric source is the fingerprint. Multiple types of information can be derived from a fingerprint, including minutiae and the pattern, such as whorl, etc. The `Finger` package contains the types, classes, and other items that are related to fingers and fingerprints. Objects of the `Finger` classes are typically not used in a stand-alone fashion, but are usually obtained from an object in the `DataInterchage` (Chapter 16 on page 51) package.

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

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

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

14.1 ANSI/NIST Minutiae Data Record

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

The BECommon provides several classes that are derived from a base View class, contained within the `Finger` package. See Chapter 14 for more information on the types associated with fingers and fingerprints. This section discusses finger views, the classes which are derived from the general `View` class. These subclasses represent specific biometric file types, such as ANSI/NIST or INCITS/M1. In the latter case, two files must be provided when constructing the object because INCITS finger image and finger minutiae records are defined in two separate standards.

14.1.1 ANSI/NIST Finger Views

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

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

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

Listing 14.1 shows how an application can use the `AN2KViewFixedResolution` to retrieve image information, image data, and derived minutiae information from a file containing an ANSI/NIST record with Type-3 (fixed resolution image) and Type-9 (fingerprint minutiae) records.

Listing 14.1: Using an AN2K Finger View

```
#include <fstream>
#include <iostream>
#include <be_finger_an2kview_fixedres.h>
using namespace std;
using namespace BiometricEvaluation;

int
main(int argc, char* argv[]) {

    Finger::AN2KViewFixedResolution *_an2kv
    try {
        _an2kv = new Finger::AN2KViewFixedResolution("type9-3.an2k",
            TYPE_3_ID, 1);
    } catch (Error::DataError &e) {
        cerr << "Caught " << e.getInfo()  << endl;
        return (EXIT_FAILURE);
    } catch (Error::FileError& e) {
        cerr << "A file error occurred: " << e.getInfo() << endl;
        return (EXIT_FAILURE);
    }
    std::auto_ptr<Finger::AN2KView> an2kv(_an2kv);

    cout << "Image resolution is " << an2kv->getImageResolution() << endl;
    cout << "Image size is " << an2kv->getImageSize() << endl;
    cout << "Image depth is " << an2kv->getImageDepth() << endl;
    cout << "Compression is " << an2kv->getCompressionAlgorithm() << endl;
    cout << "Scan resolution is " << an2kv->getScanResolution() << endl;

    // Save the finger image to a file.
    tr1::shared_ptr<Image::Image> img = an2kv->getImage();
    if (img.get() == NULL) {
       cerr << "Image was not present." << endl;
       return (EXIT_FAILURE);
    }
    string filename = "rawimg";
    ofstream img_out(filename.c_str(), ofstream::binary);
    img_out.write((char *)&(img->getRawData()[0]),
        img->getRawData().size());
    if (img_out.good())
            cout << "\tFile: " << filename << endl;
    else {
        img_out.close();
        cerr << "Error occurred when writing " << filename << endl;
        return (EXIT_FAILURE);
    }
```

```
46      img_out.close();
47
48      // Get the finger minutiae sets. AN2K records can have more than one
49      // set of minutiae for a finger.
50
51      vector<Finger::AN2KMinutiaeDataRecord> mindata = an2kv->getMinutiaeDataRecordSet();
52  }
```

14.1.2 ISO/INCITS Finger Views

The ISO [13] and INCITS [12] standards typically use separate files for the source biometric data and the derived data. For example, the ISO 19794-2 standard is for fingerprint minutiae data, while 19794-4 is for finger image data. The corresponding BECommon view objects are constructed with both files, although a view can be constructed with only one file. In the latter case, the view object will represent only that information contained in the single file.

Listing 14.1 on the preceding page shows how an application can create a view from a ANSI/INCTIS 378 finger minutiae format record [1].

Listing 14.2: Using an INCITS Finger View

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

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

Chapter 15

View

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

In the case where a raw image is part of the biometric record, the `View` object's related `Image` (Chapter 10 on page 33 object will have identical size, resolution, etc. values because the `View` class sets the `Image` attributes directly. For other image types (e.g. `JPEG`) the `Image` object will return attribute values taken from the image data.

`View` s are high-level abstractions of the biometric sample, and concrete implementations of a `View` include finger, face, iris, etc. views based on a specific type of biometric. Therefore, `View` objects are not created directly, Subclasses, such as finger views (see Chapter 14 on page 45), represent the specific type of biometric sample.

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

Chapter 16

Data Interchange

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

The design of classes in the DataInterchange package allows applications to create a single object from a biometric record, such as an ANSI/NIST file. After creating this object, the application can retrieve the needed information (such as Finger View s Chapter 14 on page 45) from this object. A typical example would be to retrieve all images from the record and pass them into a function that extracts a biometric template or some other image processing.

16.1 ANSI/NIST Data Records

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

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

Listing 16.1 shows how an application can retrieve all finger captures (Type-4 records) from an ANSI/NIST record. Once the Views are retrieved, the application obtains the set of minutiae records associated with that View.

Listing 16.1: Retrieving ANSI/NIST Finger Captures

```
1  #include <iostream>
2  #include <be_error_exception.h>
3  #include <be_finger_an2kview_capture.h>
4
5  int
6  main(int argc, char* argv[])
7  {
8      /*
9       * Call the constructor that will open an existing AN2K file and
10      * retrieve the first finger capture (Type-14) record.
11      */
```

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

```
68      }
69      cout << "There are " << cps.size() << " cores." << endl;
70      cout << "There are " << dps.size() << " deltas." << endl;
71
72      cout << "Fingerprint Reader: " << endl;
73      try { cout << an2k7m->getOriginatingFingerprintReadingSystem() << endl; }
74      catch (Error::ObjectDoesNotExist) { cout << "<Omitted>" << endl; }
75
76      cout << "Pattern (primary): " <<
77      Feature::AN2K7Minutiae::convertPatternClassification(
78      an2k7m->getPatternClassificationSet().at(0)) << endl;
79
80      return(EXIT_SUCCESS);
81  }
```

Listing 16.2 shows how an application can retrieve all latent finger images from a set of ANSI/NIST record retrieved from a `RecordStore`. Using the `Image` object, the image's "raw" data can be retrieved and passed to another function for processing. Note that the image data may be stored in a compressed format inside the ANSI/NIST record, but is converted to raw format by the `Image` object.

Listing 16.2: Retrieving ANSI/NIST Latent Records

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

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

16.2 INCITS Data Records

This INCITS class of data records covers all those record formats that are derived from the standards defined by the InterNational Committee for Information Technology Standards [12]. These formats include the ANSI-2004 Finger Minutiae Record Format [1], the ISO equivalent [2], and other data formats, including finger images.

16.2.1 Finger Views

Within the BECommon, finger view objects (Section 15) can be created from a combination of finger minutiae and image records. However, it is not necessary to have both records in order to create the view because each record contains enough information to represent the finger (image size, for example). However, if a view is contructed using only the minutiae record, then the image itself will not be present. Alternatively, if a view is made from an image record, no minutiae data would be available. It is possible to construct a view without any information.

Listing 14.2 on page 47 shows an example of accessing the information in an ANSI 378-2004 Finger Minutiae Record by creating an `ANSI2004View` object from the record file.

Chapter 17

Messaging

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

17.1 Message Center

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

Listing 17.1: Basic `MessageCenter` Usage

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

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

17.2 Command Center

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

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

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

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

Listing 17.2: Basic `CommandCenter` Usage

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

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

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

Chapter 18

Parallel Processing

18.1 MPI Parallel Processing Package

The `MPI` package is a set of APIs used implement parallel processing using the MPI [15] network-based messaging system. The core concept implemented in the framework is that of a distributor, one or more receivers, work packages, and a processing element to be implemented by the application.

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

Figure 18.1 on the following page shows the processes and data flow for a typical parallel job using components of the Framework. The distributor process executes code from the `Distributor` class, and the receiver process likewise executes `Receiver` class code. Within each process is shown the Framework packages that could be used for the job. The *Lib* element refers to the "black-box" component of software being tested, a fingerprint matching library, for example. In this example, a record store is used as the data source, and record keys are sent in the work packages. On the receiving side, the keys are used to read record data (values) from the same store.

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

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

Second, each Task-N can perform some work before creating the workers. One example is the loading of a large data set into memory; again, this is done within the application. Because Task-N calls the POSIX function `fork()` to create the workers, each worker inherits the work done by Task-N. In the case of a memory load, each worker now has that memory mapped into it's address space. See Section 18.5 on page 61 for more details.

18.2 Work Package

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

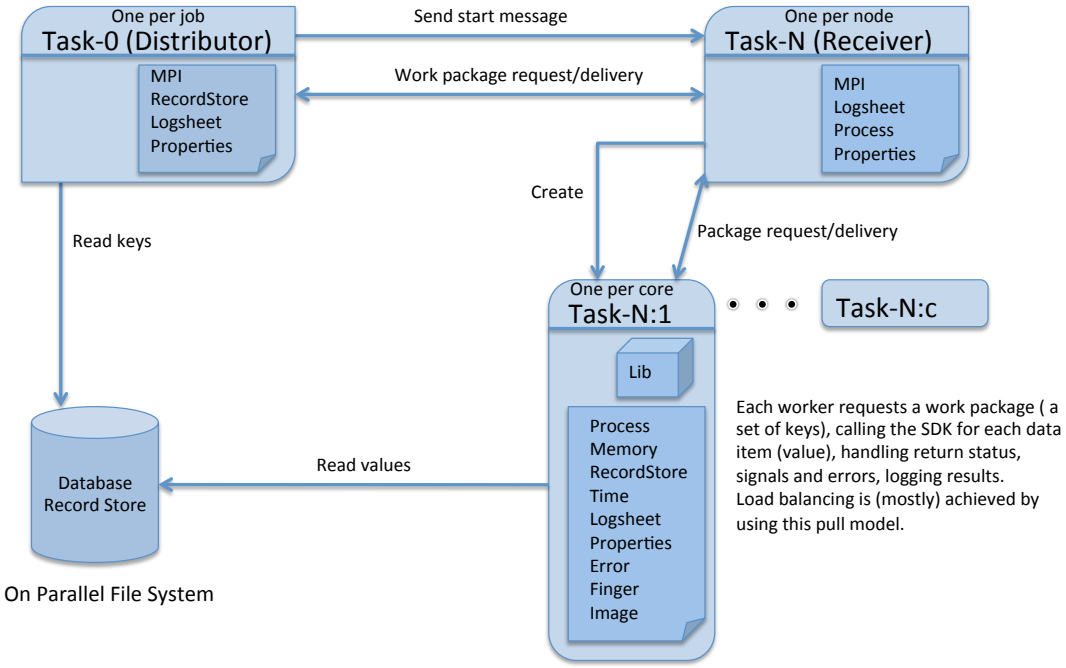


Figure 18.1: MPI Parallel Job Processes and Data Flow

The classes `RecordStoreDistributor` (Section 18.3.1) and `RecordProcessor` (Section 18.5.1 on the next page) are examples of `WorkPackage` clients that insert and remove data from a work package.

18.3 Distributor

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

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

18.3.1 Record Store Distributor

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

An application can create an instance of a `RecordStoreDistributor` with the name of a record store in order to distribute records for processing across the MPI job. Listing 18.3 on page 68 shows an example section of code to create a record store distributor. In this type of application there is no need for the application code to refine any of the Framework classes.

18.4 Receiver

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

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

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

18.5 Work Package Processor

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

Implementations of this class can be considered to have dual responsibilities. First is the management of common state used by all workers (Task-N:c in Figure 18.1 on the preceding page); creating state data shared by all workers, for example. Second, as a factory to create a package consumer for the worker process.

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

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

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

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

18.5.1 Record Processor

`RecordProcessor` is a partial implementation of `WorkPackageProcessor` and defines the `processWorkPackage()` of the `WorkPackageProcessor` interface; other methods are declared as pure-virtual and must be implemented by a child class. In addition, `RecordProcessor` declares a new pure-virtual method, `processRecord()` to be implemented by a subclass to process a single record from the record store. In summary, `RecordProcessor` removes records from the work package to be processed within the subclass, which is defined by the application. See Listing 18.1 on the facing page and Listing 18.2 on page 64 for a example of such an implementation.

18.6 MPI Resources

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

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

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

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

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

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

Input Record Store The input record store;

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

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

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

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

18.7 MPI Runtime

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

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

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

SIGQUIT Exit when the current work package is exhausted;

SIGINT Exit when the current work item is finished ("quick exit");

SIGTERM Exit immediately ("termination exit").

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

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

18.8 Logging

In order to aid tracing and debugging of a parallel job, the MPI Framework can be configured to write trace messages to the log storage. These trace messages are logged as debug messages instead of normal entries. The type and location of the log is given to the Framework by using a URL as a property when starting the MPI job (see Section 18.6 on the preceding page).

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

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

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

18.9 MPI Framework Applications

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

Listing 18.1 shows the header file that declares a subclass of `RecordProcessor`. The `newProcessor()`, `performInitialization()`, and `processRecord()` methods are those required to complete an implementation of `RecordProcessor`. A memory buffer pointer is managed with a smart pointer object.

Listing 18.1: MPI Framework Application Classes

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

```
public:
        /**
         * @brief
         * The property string ``Logsheet URL''.
         */
        static const std::string RECORDLOGSHEETURLPROPERTY;

        static const uint32_t SHAREDMEMORYSIZE = 2048;

        TestRecordProcessor(
            const std::string &propertiesFileName);
        ~TestRecordProcessor();

        std::shared_ptr<BE::MPI::WorkPackageProcessor>
        newProcessor(std::shared_ptr<BE::IO::Logsheet> &logsheet);

        void
        performInitialization(std::shared_ptr<BE::IO::Logsheet> &logsheet);

        void processRecord(const std::string &key);

        void processRecord(
            const std::string &key,
            const BE::Memory::uint8Array &value);

protected:
private:
        std::shared_ptr<BE::IO::Logsheet> _recordLogsheet;
        std::shared_ptr<char> _sharedMemory;
        uint32_t _sharedMemorySize;
};
```

Next, Listing 18.2 shows the implementation of the class methods. In this simple example, each record is acknowledged with a log entry.

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

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

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

Listing 18.2: MPI Framework Application Implementation

```
#include <be_mpi_receiver.h>
#include <be_mpi_recordstoredistributor.h>
#include <be_mpi_runtime.h>

#include "test_be_mpi.h"

using namespace BiometricEvaluation;

static const std::string DefaultPropertiesFileName("test_be_mpi.props");
```

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

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

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

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

Listing 18.3: MPI Framework Application Main

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

References

[1] *ANSI INCITS 378-2004: Finger Minutiae Format for Data Interchange*. ANSI/INCITS, 2004. 43, 47, 54

[2] *ISO/IEC 19794-2: Information technology - Biometric data interchange formats - Part 2: Finger minutiae data*. ISO/IEC, first edition, 2005. 43, 54

[3] *American National Standard for Information Systems - Data Format for the Interchange of Fingerprint, Facial, & Other Biometric Information*. ANSI/NIST-ITL, 1-2007 edition, 2007. 4, 43, 45, 51

[4] Mark Adler. zlib, 2012. http://www.zlib.net. 19

[5] Berkeley DB. https://sqlite.org. 72

[6] World Wide Web Consortium. Portable Network Graphics Standard, 2003. http://www.w3.org/TR/PNG/. 35

[7] FFmpeg Multimedia Framework. http://www.ffmpeg.org. 37, 72

[8] GNU Make Project. http://www.gnu.org/software/make. 71

[9] Independent JPEG Group. libjpeg, 2011. http://www.ijg.org/. 34

[10] Joint Photographic Experts Group. JPEG2000 Image Standard, 1992. http://www.jpeg.org/jpeg2000/index.html. 34

[11] Joint Photographic Experts Group. JPEG Image Standard, 2011. http://www.jpeg.org/jpeg/index.html. 34

[12] International Committee for Information Technology Standards. http://www.incits.org. 47, 54

[13] ISO/IEC Joint Technical Committee 1/SC 37 Biometrics. 47

[14] Communications and Remote Sensing Lab, Université catholique de Louvain. OpenJPEG Library, 2011. http://www.openjpeg.org/. 34, 72

[15] Message Passing Interface Forum. http://www.mpi-forum.org. 4, 59, 71

[16] NIST Biometric Image Software, 2011. http://www.nist.gov/itl/iad/ig/nbis.cfm. 7, 34, 35, 72

[17] NIST Image Group. http://www.nist.gov/itl/iad/ig. 1

[18] The Open MPI Project. http://www.openmpi.org. 72, 73

[19] The OpenSSL Project. http://www.openssl.org. 72

[20] Greg Roelofs. libpng, 2011. http://www.libpng.org/pub/png/libpng.html. 35, 72

[21] The SQLite Project. https://sqlite.org. 72

[22] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, special edition, 2000. 3, 9

[23] The Syslog Protocol. http://tools.ietf.org/html/rfc5424. 18

[24] Wavelet Scalar Quantization Gray-Scale Fingerprint Image Compression Standard, 2010. https://www.fbibiospecs.org/docs/WSQ_Gray-scale_Specification_Version_3_1_Final.pdf. 35

Appendix A

Building the Framework

A.1 Language Features

The Biometric Evaluation Framework was developed using the 2011 version of the C++ language standard. It is not possible to subset BECommon to use an earlier version of C++.

Two implementations of C++11 known to compile BECommon are:

- GNU Compiler Collection version 4.8.2 on Linux.
- Apple LLVM version 6.0 (clang-600.0.56) on OS-X.

A.2 The Framework Build System

The distribution of BECommon includes a set of `make` files used to build the BECommon library, as well as install the library and header files. These `make` file use some features of the GNU make [8] system, and therefore the GNU software must be installed on the user's system. Future versions of BECommon may use a different build system.

In order to tailor the build of the BECommon library (file `libbiomeval`), the `common/src/libbiomeval/Makefile` file needs editing. At the top of this file are make variables for locating the header files and libraries for NBIS, and other libraries.

The make file also sets variables that create subsets of the BECommon. `CORE` and `IO` are required as they form the basis of the BECommon. The `SOURCES` variable contains a list of variables pertaining to the desired build of BECommon.

A.3 External Software Dependencies

The Biometric Evaluation Framework is built upon several other software packages. The packages are used for image processing, biometric data record formats, the message passing interface [15], as well as operating system and compiler tool chains.

Other common software development libraries used by BECommon are documented in the sections that follow. Specific instructions for installing these packages are not given here. However, in general, many systems that provide a packaging system split the library support into two packages: One for runtime (containing the binary library file only), and one for use when developing applications. This second package installs the header files needed to build the BECommon.

A.3.1 NIST Biometric Image Software

The NIST Biometric Image Software (NBIS) [16] is a set of packages used for ANSI-NIST record processing, and other support. The Biometric Evaluation Framework uses NBIS for ANSI-NIST support, and therefore the NBIS packages need to be compiled and installed prior to building BECommon.

A.3.2 Video and Image Processing

For the `Image` classes, the OpenJPEG [14] and PNG [20] development libraries are required.

For `Video` classes, the FFmpeg [7] libraries are used. When building from source, configure to build and install shared libraries. By default, only static libraries are built.

A.3.3 Cryptography

Cryptography support is provided by the OpenSSL [19] library. An example is the `openssl-devel` package on Linux systems which provides the `libcrypto` file and associated header files for development.

A.3.4 Sqlite

SQLite is an embedded Structured Query Language (SQL) database engine and is used by the `IO::SQLiteRecordStore` class to provide an `IO::RecordStore` that is backed by a SQLite database. Information on SQLite can be found at [21].

A.3.5 Berkeley Database

The Berkeley Database BDB [5] is available as both open source and closed source commercial variants. The BECommon class `IO::DBRecordStore` uses the BDB software to store key/value pairs. There are two versions of the BDB API; BECommon uses version 1.85 as defined in the original open source distribution.

A.3.6 Message Passing Interface

An implementation of the MPI specification must be installed on the user's system before the full BECommon can be built. However, the `MPI` package can be optionally left out of the BECommon build system, if desired.

One common implementation of MPI is OpenMPI [18], available as source code, or binary packages. Often the MPI runtime is a separate binary package from the MPI development software. As an example, for many Linux distributions, an example of the runtime package is `openmpi-1.6.4-3`, while the related development package would be `openmpi-devel-1.6.4-3`.

The location of the OpenMPI libraries may be installed in a specific location. For example, on the CnetOS-7 Linux distribution, the MPI libraries are installed on `/usr/lib64/openmpi/lib/`, but the dynamic linker configuration will not locate those libraries, and linking of an application against the BECommon library will fail. To fix this problem create `/etc/ld.so.conf.d/openmpi.conf` with the line `/usr/lib64/openmpi/lib/`, then run the `ldconfig` command (as root) to update the dynamic linker configuration.

To build the BECommon, both packages are installed. In order to run an MPI job, only the runtime package needs to be installed on all nodes that participate in the MPI job. Chapter B has more information on running an MPI job.

Appendix B

Running an MPI Job

B.1 OpenMPI

This chapter describes how to use the OpenMPI [18] runtime system to execute an MPI job. Some parameters passed to the `mpirun` command are related to the notions captured in the Biometric Evaluation Framework MPI support.

B.2 Example Shell Script

Listing B.1: Example Script to run MPI

```
#
#
# Record store for the input.
#
INPUTRS=./SD29.rs

#
# Create the properties file for this run
#
# Logsheet URL is used by the framework for logging and is optional.
# Record Logsheet URL is defined and used by the application and is
# optional in the test_mpi program.
#
# An example config file for rsyslogd, listening on a non-default port:
#
#       $ModLoad imtcp
#       # Provides TCP syslog reception
#       $InputTCPServerRun 2514
#       local0.info /home/wsalamon/sandbox/rsyslog/record.log
#       local1.debug /home/wsalamon/sandbox/rsyslog/debug.log
#
PROPS=test_mpi.props
cat > $PROPS << EOF
Input Record Store = $INPUTRS
Chunk Size = 64
Workers Per Node = 8
Logsheet URL = syslog://loghost:2514
Record Logsheet URL = syslog://loghost:2514
```

```
EOF

#
# Two forms of the nodes string, one for the script to copy all
# files out, one for the mpirun command.
#
NODES="node01b node02b node03b node04b"
MPINODES="node01b,node02b,node03b,node04b"

#
# MPIPROCS must be >= 2, is the Task-N count plus one for Task-0.
#
MPIPROCS=5

#
# Set any options to the OpenMPI mpirun command. The example below will
# turn on some tracing and how processes are mapped to nodes.
#
#MPIOPTS=" --show-progress --debug-daemons --display-devel-map"

# Where the program is run. The directory must exist on all the
# nodes, and this script must be started here.
DIR=$PWD

#
# LIBS is any libraries th must coexist with the program to be run.
#
LIBS=
PROGRAM=test_mpi
CPFILES="$PROGRAM $PROPS $LIBS"

#
# The test program and dependencies must exist on all nodes, so copy
# everything to the runtime directory on all nodes. It helps to run
# an SSH agent or something similar.
#
for n in $NODES; do
        echo $n;
        scp -p $CPFILES $n:$DIR;
done

#
# Run the program as an MPI job. mpirun must be in the users path.
# The properties file name is the only parameter to the program.
#
EXECSTR="$PROGRAM $PROPS"
mpirun $MPIOPTS -H $MPINODES -np $MPIPROCS --path $DIR $EXECSTR
```

Appendix C

API Reference

Appendix D

Namespace Index

D.1 Namespace List

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

Appendix E

Hierarchical Index

E.1 Class Hierarchy

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

Appendix F

Class Index

F.1 Class List

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

Appendix G

Namespace Documentation

G.1 BiometricEvaluation Namespace Reference

Namespaces

- Error

  Exceptions, and other error handling.
- Face

  Biometric information relating to face images and derived information.
- Feature

  Definition of an MPEG4 Face feature point. See ISO/IEC 14496-2.
- Finger

  Biometric information relating to finger images and derived information.
- Framework

  Information about the framework.
- Image

  Basic information relating to images.
- IO

  Input/Output functionality.
- Iris

  Biometric information relating to iris images and derived information.
- Memory

  Support for memory-related operations.
- MPI

  Common declarations and functions for the MPI-based functionality.
- Process

  Process information and controls.
- System

  Operating system, hardware, etc.
- Text

  Text processing for string objects.
- Time

  Support for time and timers.

- Video

  Basic information relating to video and streams.
- View

  View information.

G.1.1 Detailed Description

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

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

G.2 BiometricEvaluation::Error Namespace Reference

Exceptions, and other error handling.

Classes

- class ConversionError

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

  Error when reading data from an external source.
- class Exception

  The parent class of all BiometricEvaluation exceptions.
- class FileError

  File error when opening, reading, writing, etc.
- class MemoryError

  An error occurred when allocating an object.
- class NotImplemented

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

  The named object does not exist.
- class ObjectExists

  The named object exists and will not be replaced.
- class ObjectIsClosed

  The object is closed.
- class ObjectIsOpen

  The object is already opened.
- class ParameterError

An invalid parameter was passed to a constructor or method.

- class SignalManager

  A SignalManager object is used to handle signals that come from the operating system.

- class StrategyError

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

Functions

- std::string errorStr (bool includeErrno=false)

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

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

G.2.1 Detailed Description

Exceptions, and other error handling.

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

G.2.2 Function Documentation

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

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

Parameters

includeErrno	Whether or not to include the value of errno in the returned string.

Returns

The current error message specified by errno.

G.3 BiometricEvaluation::Face Namespace Reference

Biometric information relating to face images and derived information.

Classes

- class INCITSView

  A class to represent single facial image view and derived information.

- class ISO2005View

  A class to represent single face view and derived information.

- struct PoseAngle

  Representation of pose angle and uncertainty.

Typedefs

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

Enumerations

- enum Gender { **Unspecified** = 0x00, **Male** = 0x01, **Female** = 0x02, **Unknown** = 0xFF }

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

  Eye color.
- enum HairColor {
  Unspecified = 0x00, **Bald** = 0x01, **Black** = 0x02, **Blonde** = 0x03,
  Brown = 0x04, **Gray** = 0x05, **White** = 0x06, **Red** = 0x07,
  Unknown = 0xFF }

  Hair color.
- enum Property {
  Glasses = 1, **Moustache** = 2, **Beard** = 3, **Teeth** = 4,
  Blink = 5, **MouthOpen** = 6, **LeftEyePatch** = 7, **RightEyePatch** = 8,
  DarkGlasses = 9, **MedicalCondition** = 10 }

  Face property codes.
- enum Expression {
  Unspecified = 0x0000, **Neutral** = 0x0001, **SmileClosedJaw** = 0x0002, **SmileOpenJaw** = 0x0003,
  RaisedEyebrows = 0x0004, **EyesLookingAway** = 0x0005, **Squinting** = 0x0006, **Frowning** = 0x0007 }

  Face expression codes.
- enum ImageType { **Basic** = 0x00, **FullFrontal** = 0x01, **TokenFrontal** = 0x02 }

  Face image type classification codes.
- enum ImageDataType { **JPEG** = 0x00, **JPEG2000** = 0x01 }

  Face image data type classification codes.
- enum ColorSpace {
  Unspecified = 0x00, **RGB24** = 0x01, **YUV422** = 0x02, **Grayscale8** = 0x03,
  Other = 0x04 }

  Color space codes.
- enum SourceType {
  Unspecified = 0x00, **StaticPhotoUnknown** = 0x01, **StaticPhotoDigitalStill** = 0x02, **StaticPhotoScan** = 0x03,
  VideoFrameUnknown = 0x04, **VideoFrameAnalog** = 0x05, **VideoFrameDigital** = 0x06, **Unknown** = 0x07 }

  Source type codes.

G.3.1 Detailed Description

Biometric information relating to face images and derived information.

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

G.3.2 Typedef Documentation

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

A set of properties.

G.4 BiometricEvaluation::Feature Namespace Reference

Definition of an MPEG4 Face feature point. See ISO/IEC 14496-2.

Classes

- class AN2K7Minutiae

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

  Representation of the core.
- struct DeltaPoint

  Representation of the delta.
- class INCITSMinutiae

  A class to represent a set of minutiae in an ANSI/INCITS record.
- class Minutiae

  A class to represent a set of minutiae data points.
- struct MinutiaPoint

  Representation of a finger minutiae data point.
- struct MPEGFacePoint

  Representation of a feature point and a set of points.
- struct RidgeCountItem

  Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number.

Typedefs

- using **AN2K7MinutiaeSet** = std::vector< std::shared_ptr< AN2K7Minutiae >>
- using **MinutiaPoint** = struct MinutiaPoint
- using **MinutiaPointSet** = std::vector< MinutiaPoint >
- using **RidgeCountItem** = struct RidgeCountItem
- using **RidgeCountItemSet** = std::vector< RidgeCountItem >
- using **CorePoint** = struct CorePoint
- using **CorePointSet** = std::vector< CorePoint >
- using **DeltaPoint** = struct DeltaPoint
- using **DeltaPointSet** = std::vector< DeltaPoint >
- using **MinutiaeSet** = std::vector< std::shared_ptr< Minutiae >>
- typedef std::vector
  < MPEGFacePoint > **MPEGFacePointSet**

Enumerations

- enum MinutiaeFormat {
  AN2K7 = 0, **IAFIS**, **Cogent**, **Motorola**,
  Sagem, **NEC**, **Identix**, **M1** }

  Enumerate the minutiae format standards.
- enum MinutiaeType { **RidgeEnding** = 0, **Bifurcation**, **Compound**, **Other** }

  Enumerate the types of minutiae: Ridge Ending, Bifurcation, Compound, or other.
- enum RidgeCountExtractionMethod { **NonSpecific** = 0, **FourNeighbor** = 1, **EightNeighbor** = 2, **Other** = 3 }

  Enumerate the types of extraction methods for ridge counts.

Functions

- std::ostream & operator<< (std::ostream &, const AN2K7Minutiae::FingerprintReadingSystem &)

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

G.4.1 Detailed Description

Definition of an MPEG4 Face feature point. See ISO/IEC 14496-2.

G.5 BiometricEvaluation::Finger Namespace Reference

Biometric information relating to finger images and derived information.

Classes

- class AN2KMinutiaeDataRecord

  Representation of a Type-9 Record from an AN2K file.
- class AN2KView

  A class to represent single finger view and derived information.
- class AN2KViewCapture

  Represents an ANSI/NIST variable-resolution finger image.
- class AN2KViewFixedResolution

  A class to represent single finger view and derived information.
- class AN2KViewLatent
- class AN2KViewVariableResolution

  A class to represent single finger view based on an ANSI/NIST record.
- class ANSI2004View

  A class to represent single finger view and derived information.
- class ANSI2007View

  A class to represent single finger view and derived information.
- class INCITSView

  A class to represent single finger view and derived information.
- class ISO2005View

  A class to represent single finger view and derived information.

Typedefs

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

Enumerations

- enum PatternClassification {
  PlainArch = 0, **TentedArch**, **RadialLoop**, **UlnarLoop**,
  PlainWhorl, **CentralPocketLoop**, **DoubleLoop**, **AccidentalWhorl**,
  Whorl, **RightSlantLoop**, **LeftSlantLoop**, **Scar**,
  Amputation, **Unknown** }
- enum Position {
  Unknown = 0, **RightThumb** = 1, **RightIndex** = 2, **RightMiddle** = 3,
  RightRing = 4, **RightLittle** = 5, **LeftThumb** = 6, **LeftIndex** = 7,
  LeftMiddle = 8, **LeftRing** = 9, **LeftLittle** = 10, **PlainRightThumb** = 11,
  PlainLeftThumb = 12, **PlainRightFourFingers** = 13, **PlainLeftFourFingers** = 14, **LeftRightThumbs** = 15,
  EJI = 19 }

  Finger position codes.
- enum Impression {
  LiveScanPlain = 0, **LiveScanRolled**, **NonLiveScanPlain**, **NonLiveScanRolled**,
  LatentImpression, **LatentTracing**, **LatentPhoto**, **LatentLift**,
  LiveScanVerticalSwipe, **LiveScanPalm**, **NonLiveScanPalm**, **LatentPalmImpression**,
  LatentPalmTracing, **LatentPalmPhoto**, **LatentPalmLift**, **LiveScanOpticalContactPlain**,
  LiveScanOpticalContactRolled, **LiveScanNonOpticalContactPlain**, **LiveScanNonOpticalContact↩ Rolled**, **LiveScanOpticalContactlessPlain**,
  LiveScanOpticalContactlessRolled, **LiveScanNonOpticalContactlessPlain**, **LiveScanNonOptical↩ ContactlessRolled**, **Other**,
  Unknown }
- enum FingerImageCode {
  EJI = 0, **RolledTip**, **FullFingerRolled**, **FullFingerPlainLeft**,
  FullFingerPlainCenter, **FullFingerPlainRight**, **ProximalSegment**, **DistalSegment**,
  MedialSegment, **NA** }

Functions

- std::ostream & operator<< (std::ostream &stream, const AN2KViewCapture::FingerSegmentPosition &fsp)

  Output stream overload for FingerSegmentPosition.
- std::ostream & operator<< (std::ostream &stream, const AN2KViewVariableResolution::PrintPosition↩ Coordinate &ppc)

  Output stream overload for PrintPositionCoordinate.

G.5.1 Detailed Description

Biometric information relating to finger images and derived information.

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

G.5.2 Enumeration Type Documentation

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

Joint and tip codes.

enum BiometricEvaluation::Finger::Impression [strong]

Finger and palm impression types.

enum BiometricEvaluation::Finger::PatternClassification [strong]

Pattern classification codes.

enum BiometricEvaluation::Finger::Position [strong]

Finger position codes.

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

G.5.3 Function Documentation

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

Output stream overload for PrintPositionCoordinate.

Parameters

in	*stream*	Stream on which to append formatted PrintPositionCoordinate information.
in	*ppc*	PrintPositionCoordinate information to append to stream.

Returns

Stream with a ppc textual representation appended.

G.6 BiometricEvaluation::Framework Namespace Reference

Information about the framework.

Classes

- class ConstEnumMapWrapper

  Wrapper class around an individual enumeration entity (const).
- class EnumerationFunctions
- class EnumMapWrapper

  Wrapper class around an individual enumeration entity (non-const).

Functions

- unsigned int getMajorVersion ()

  Framework major version.
- unsigned int getMinorVersion ()

  Framework minor version.
- std::string getCompiler ()

  Compiler used to compile this framework.
- std::string getCompileDate ()

  Date when this framework was compiled.
- std::string getCompileTime ()

Time when this framework was compiled.

- std::string getCompilerVersion ()

  Version string of compiler used to compile this framework.

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

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

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

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

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

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

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

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

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

  Append the string representation of an enumeration into a stream.

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

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

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

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

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

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

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

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

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

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

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

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

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

  Append the string representation of an enumeration into a stream.

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

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

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

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

G.6.1 Detailed Description

Information about the framework.

G.6.2 Function Documentation

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

Date when this framework was compiled.

Returns

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

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

Compiler used to compile this framework.

Returns

The name of the compiler used to compile this framework.

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

Version string of compiler used to compile this framework.

Returns

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

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

Time when this framework was compiled.

Returns

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

unsigned int BiometricEvaluation::Framework::getMajorVersion ( )

Framework major version.

Returns

The major version number of the BiometricFramework

unsigned int BiometricEvaluation::Framework::getMinorVersion ( )

Framework minor version.

Returns

The minor version of the BiometricEvaluation framework.

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

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

Parameters

lhs	The string to compare to the enumeration.
rhs	The enumeration to compare to the string.

Returns

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

Note

String comparison is case-sensitive.

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

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

lhs	The enumeration to compare to the string.
rhs	The string to compare to the enumeration.

Returns

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

Note

String comparison is case-sensitive.

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

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

lhs	The string to compare to the enumeration.
rhs	The enumeration to compare to the string.

Returns

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

Note

String comparison is case-sensitive.

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

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

Parameters

lhs	The enumeration to compare to the string.
rhs	The string to compare to the enumeration.

Returns

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

Note

String comparison is case-sensitive.

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

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

lhs	Existing string.
rhs	Enumeration whose string representation should be concatenated.

Returns

String made by appending string representation of rhs to lhs.

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

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

lhs	Enumeration whose string representation should be concatenated.
rhs	Existing string.

Returns

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

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

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

lhs	Existing string.
rhs	Enumeration whose string representation should be concatenated.

Returns

String made by appending string representation of rhs to lhs.

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

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

Parameters

lhs	Enumeration whose string representation should be concatenated.
rhs	Existing string.

Returns

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

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

Append the string representation of an enumeration into a stream.
Parameters

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

Returns

Reference to stream.

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

Append the string representation of an enumeration into a stream.
Parameters

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

Returns

Reference to stream.

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

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

lhs	The string to compare to the enumeration.
rhs	The enumeration to compare to the string.

Returns

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

Note

String comparison is case-sensitive.

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

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

Parameters

lhs	The enumeration to compare to the string.
rhs	The string to compare to the enumeration.

Returns

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

Note

String comparison is case-sensitive.

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

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

lhs	The string to compare to the enumeration.
rhs	The enumeration to compare to the string.

Returns

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

Note

String comparison is case-sensitive.

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

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

lhs	The enumeration to compare to the string.
rhs	The string to compare to the enumeration.

Returns

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

Note

String comparison is case-sensitive.

G.7 BiometricEvaluation::Image Namespace Reference

Basic information relating to images.

Classes

- class BMP

  A BMP-encoded image.
- struct Coordinate

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

  Represent attributes common to all images.
- class JPEG

  A JPEG-encoded image.
- class JPEG2000

  A JPEG-2000-encoded image.
- class JPEGL

  A Lossless JPEG-encoded image.
- class NetPBM

  A NetPBM-encoded image.
- class PNG

  A PNG-encoded image.
- class Raw

  An image with no encoding or compression.
- struct Resolution

  A structure to represent the resolution of an image.
- struct Size

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

  A WSQ-encoded image.

Typedefs

- using **Coordinate** = struct Coordinate
- using **CoordinateSet** = std::vector< Image::Coordinate >
- using **Size** = struct Size
- using **Resolution** = struct Resolution

Enumerations

- enum CompressionAlgorithm {
  None = 0, **Facsimile** = 1, **WSQ20** = 2, **JPEGB** = 3,
  JPEGL = 4, **JP2** = 5, **JP2L** = 6, **PNG** = 7,
  NetPBM = 8, **BMP** = 9 }
- enum PixelFormat { PixelFormat::MonoWhite = 0, PixelFormat::MonoBlack = 1, PixelFormat::Gray8 = 2, PixelFormat::RGB24 = 3 }

Functions

- std::ostream & **operator**<< (std::ostream &, const Coordinate &)
- std::ostream & operator<< (std::ostream &stream, const CoordinateSet &coordinates)

  Output stream overload for CoordinateSet.
- std::ostream & **operator**<< (std::ostream &, const Size &)
- std::ostream & **operator**<< (std::ostream &, const Resolution &)
- float distance (const Coordinate &p1, const Coordinate &p2)

  Calculate the distance between two points.

G.7.1 Detailed Description

Basic information relating to images.

Classes and methods for manipulating images.

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

G.7.2 Enumeration Type Documentation

enum BiometricEvaluation::Image::CompressionAlgorithm [strong]

Image compression algorithms.

enum BiometricEvaluation::Image::PixelFormat [strong]

Image pixel formats.

Enumerator

MonoWhite 1 bit/pixel, 0 is white, 1 = black

MonoBlack 1 bit/pixel, 0 is black, 1 = white

Gray8 8-bit gray

RGB24 8-bit red/8-bit blue/8-bit green

G.7.3 Function Documentation

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

Calculate the distance between two points.

Parameters

in	*p1*	First point.
in	*p2*	Second point.

Returns

Distance between p1 and p2.

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

Output stream overload for CoordinateSet.

Parameters

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

Returns

stream with a coordinates textual representation appended.

G.8 BiometricEvaluation::IO Namespace Reference

Input/Output functionality.

Namespaces

- Utility

Classes

- class ArchiveRecordStore

  This class implements the IO::RecordStore interface by storing data items in single file, with an associated manifest file.
- class CompressedRecordStore

  Sibling-implemented RecordStore with Compression.
- class Compressor
- class DBRecordStore

  A class that implements IO::RecordStore using a Berkeley DB database as the underlying record storage system.
- class FileLogCabinet
- class FileLogsheet

  A class to represent a single logging mechanism with a file as the backing store.
- class FileRecordStore
- class GZip

  Compressor for gzip compression from zlib.
- class ListRecordStore

  RecordStore that reads a list of keys from a text file, and retrieves the data from another RecordStore.
- class Logsheet

  A class to represent a logging mechanism.
- struct ManifestEntry
- class Properties

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

  A Properties object persisted in an file on disk.
- class RecordStore

  A class to represent a data storage mechanism.
- class RecordStoreIterator

  Generic ForwardIterator for all RecordStores.
- class SQLiteRecordStore

  A RecordStore implementation using a SQLite database as the underlying record storage system.
- class SysLogsheet

  A class to represent a single logging mechanism to a logging service on the network.

Typedefs

- using **ManifestEntry** = struct ManifestEntry
- using ManifestMap = Memory::OrderedMap< std::string, ManifestEntry >

G.8.1 Detailed Description

Input/Output functionality.

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

G.8.2 Typedef Documentation

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

Convenience alias for storing the manifest

G.9 BiometricEvaluation::IO::Utility Namespace Reference

Functions

- void removeDirectory (const std::string &directory, const std::string &prefix)

  Remove a directory using directory name and parent pathname.
- void removeDirectory (const std::string &pathname)

  Remove a directory using a complete pathname.
- void copyDirectoryContents (const std::string &sourcepath, const std::string &targetpath, const bool removesource=false)

  Copy the contents of a directory, optionally deleting the source directory contents when done.
- void setAsideName (const std::string &name)

  Set aside a file or directory name.
- uint64_t getFileSize (const std::string &pathname)
- uint64_t sumDirectoryUsage (const std::string &pathname)
- bool fileExists (const std::string &pathname)
- bool **pathIsDirectory** (const std::string &pathname)
- int makePath (const std::string &path, const mode_t mode)

  Create an entire directory tree.
- Memory::uint8Array readFile (const std::string &path, std::ios_base::openmode mode=std::ios_base::binary)

  Read the contents of a file into a buffer.
- void writeFile (const uint8_t *data, const size_t size, const std::string &path, std::ios_base::openmode mode=std::ios_base::binary)

  Write the contents of a buffer to a file.
- void writeFile (const Memory::uint8Array data, const std::string &path, std::ios_base::openmode mode=std::ios_base::binary)

  Write the contents of a buffer to a file.
- bool isReadable (const std::string &pathname)

  Determine if the real user has read access permissions to this file.
- bool isWritable (const std::string &pathname)

Determine if the real user has write access permissions to this file.

- std::string createTemporaryFile (const std::string &prefix="", const std::string &parentDir="/tmp")

  Create a temporary file.

- FILE * createTemporaryFile (std::string &path, const std::string &prefix="", const std::string &parentDir="/tmp")

  Create a temporary file.

G.9.1 Detailed Description

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

G.9.2 Function Documentation

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

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

Parameters

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

Exceptions

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

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

Create a temporary file.

Parameters

in	*prefix*	String to be prefixed to the random temporary name.
in	*parentDir*	Where to place the temporary file.

Exceptions

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

Returns

Path to temporary file.

Note

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

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

Create a temporary file.

Exclusivity to the file stream is guaranteed.

Parameters

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

Exceptions

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

Returns

Open file stream to path.

Note

Caller must fclose(3) the returned stream.

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

Indicate whether a file exists.

Parameters

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

Returns

true if the file exists, false otherwise.

Exceptions

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

uint64_t BiometricEvaluation::IO::Utility::getFileSize (const std::string & *pathname*)

Get the size of a file.

Parameters

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

Returns

The file size.

Exceptions

Error::ObjectDoesNot↩ Exist	The named directory does not exist.
Error::StrategyError	An error occurred when using the underlying storage system, or pathname is malformed.

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

Determine if the real user has read access permissions to this file.

Parameters

in	*pathname*	Path to the file to check.

Returns

true if the real user has read acccess permissions, false otherwise.

Warning

This function should **only** be called *after* failing to open a file, to determine a possible failure reason.

See also

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

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

Determine if the real user has write access permissions to this file.

Parameters

in	*pathname*	Path to the file to check.

Returns

true if the real user has write acccess permissions, false otherwise.

Warning

This function should **only** be called *after* failing to write to a file, to determine a possible failure reason.

See also

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

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

Create an entire directory tree.

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

Parameters

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

Returns

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

Memory::uint8Array BiometricEvaluation::IO::Utility::readFile (const std::string & *path*, std::ios_base::openmode *mode* = `std::ios_base::binary`)

Read the contents of a file into a buffer.
Parameters

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

Returns

Contents of path in a buffer.

Exceptions

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

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

Remove a directory using directory name and parent pathname.
Parameters

in	*directory*	The name of the directory to be removed, without a preceding path.
in	*prefix*	The path leading to the directory.

Exceptions

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

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

Remove a directory using a complete pathname.
Parameters

in	*pathname*	The complelte path name of the directory to be removed,

Exceptions

Error::ObjectDoesNot↩ Exist	The named directory does not exist.
Error::StrategyError	An error occurred when using the underlying storage system, or the path name is malformed.

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

Set aside a file or directory name.

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

Parameters

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

Exceptions

Error::ObjectDoesNot↩ Exist	The named object does not exist.
Error::StrategyError	An error occurred when using the underlying storage system, the name or prefix is malformed, or the maximum number of attempts was reached.

uint64_t BiometricEvaluation::IO::Utility::sumDirectoryUsage (const std::string & *pathname*)

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

Parameters

in	*pathname*	The name of the directory to be sized.

Returns

The sum of file and directory sizes.

Exceptions

Error::ObjectDoesNot↩ Exist	The named directory does not exist.
Error::StrategyError	An error occurred when using the underlying storage system, or pathname is malformed.

void BiometricEvaluation::IO::Utility::writeFile (const uint8_t * *data*, const size_t *size*, const std::string & *path*, std::ios_base::openmode *mode* = `std::ios_base::binary`)

Write the contents of a buffer to a file.

A thin wrapper around std::ofstream. The mode parameter has the same semantics as that for std::ofstream and applications must set mode for append or truncate when writing to an existing file.

Parameters

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

Exceptions

ObjectExists	path exists and is a directory.
StrategyError	An error occurred when using the underlying storage system.

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

Write the contents of a buffer to a file.

A thin wrapper around std::ofstream. The mode parameter has the same semantics as that for std::ofstream and applications must set mode for append or truncate when writing to an existing file.

Parameters

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

Exceptions

ObjectExists	path exists and is a directory.
StrategyError	An error occurred when using the underlying storage system.

G.10 BiometricEvaluation::Iris Namespace Reference

Biometric information relating to iris images and derived information.

Classes

- class INCITSView

  A class to represent single iris view and derived information.
- class ISO2011View

  A class to represent single iris view and derived information.

Enumerations

- enum CaptureDeviceTechnology { **Unknown** = 0, **CMOSCCD** = 1 }

  Capture device technology identifiers.
- enum EyeLabel { **Undefined** = 0, **Right** = 1, **Left** = 2 }

  Eye label.
- enum ImageType { **Uncropped** = 1, **VGA** = 2, **Cropped** = 3, **CroppedMasked** = 7 }

  Iris image type classification codes.
- enum Orientation { **Undefined** = 0, **Base** = 1, **Flipped** = 2 }

  Iris horizontal orientation classification codes.
- enum ImageCompression { **Undefined** = 0, **LosslessNone** = 1, **Lossy** = 2 }

Iris image compression type.

- enum CameraRange { **Unassigned** = 0, **Failed** = 1, **Overflow** = 2 }

  Range from camera lens center to subject iris.

G.10.1 Detailed Description

Biometric information relating to iris images and derived information.

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

G.11 BiometricEvaluation::Memory Namespace Reference

Support for memory-related operations.

Namespaces

- AutoArrayUtility

Classes

- class AutoArray

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

  RandomAccessIterator for any AutoArray.
- class AutoBuffer
- class IndexedBuffer

  Wrap a memory buffer with an index.
- class MutableIndexedBuffer
- class OrderedMap
- class OrderedMapConstIterator
- class OrderedMapIterator

Typedefs

- using **uint8Array** = AutoArray< uint8_t >
- using **uint16Array** = AutoArray< uint16_t >
- using **uint32Array** = AutoArray< uint32_t >

G.11.1 Detailed Description

Support for memory-related operations.

The Memory package contains templates and classes that are used to manage memory, auto-sizing arrays, for example.

G.12 BiometricEvaluation::Memory::AutoArrayUtility Namespace Reference

Functions

- template<typename T , typename = typename std::enable_if<std::is_same<T, uint8_t>::value || std::is_same<T, char>::value>::type>
  char * cstr (const AutoArray< T > &rahc)

  Cast an AutoArray of uint8_t or char to a char.*
- template<typename T , typename = typename std::enable_if<std::is_same<T, uint8_t>::value || std::is_same<T, char>::value>::type>
  std::string getString (const AutoArray< T > &aa, typename AutoArray< T >::size_type count)

  Convert a uint8_t or char AutoArray to a string.
- template<typename T , typename = typename std::enable_if<std::is_same<T, uint8_t>::value || std::is_same<T, char>::value>::type>
  void setString (AutoArray< T > &aa, const std::string &str)

  Copy a string into an AutoAray of uint8_t or char.
- template<typename T , typename = typename std::enable_if<std::is_same<T, uint8_t>::value || std::is_same<T, char>::value>::type>
  void setString (AutoArray< T > &aa, const char *str,...)

  Copy a string into an AutoAray of uint8_t or char.

G.12.1 Detailed Description

Convenience functions for AutoArrays.

G.12.2 Function Documentation

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

Cast an AutoArray of uint8_t or char to a char*.

Parameters

rahc	AutoArray to cast.

Returns

rahc casted as a char*.

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

Convert a uint8_t or char AutoArray to a string.

Parameters

aa	AutoArray to stringify.
count	Last byte of aa to include in the returned string.

Returns

String representation of aa.

Exceptions

Error::MemoryError	count > aa.size()

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

Copy a string into an AutoAray of uint8_t or char.
Parameters

aa	AutoArray whose contents will be replaced with str.
str	String to assign to AutoArray.

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

Copy a string into an AutoAray of uint8_t or char.
Parameters

aa	AutoArray whose contents will be replaced with str.
str	printf-style format string.
...	Variable list of arguments for printf formatting.

G.13 BiometricEvaluation::MPI Namespace Reference

Common declarations and functions for the MPI-based functionality.

Classes

- class Distributor

  A class to represent an MPI task that distributes work to other tasks.
- class MessageTag

  The types of messages sent between MPI task processes.
- class Receiver

  A class to represent an MPI task that receives WorkPackages containers from the Distributor.
- class RecordProcessor

  An implementation of a work package processor that will extract record store keys, and optionally, values, from a WorkPackage.
- class RecordStoreDistributor

An implementation of the Distrbutor abstraction that uses a record store for input to create the work packages.

- class RecordStoreResources

  A class to represent a set of resources needed by an MPI program using a RecordStore for input.
- class Resources
- class Runtime

  Runtime support for the startup/shutdown of MPI jobs.
- class TaskCommand

  The command given to an MPI task.
- class TaskStatus

  The status of an MPI distributor or receiver task.
- class WorkPackage

  A class to represent a piece of work to be acted upon by a processor.
- class WorkPackageProcessor

  Represents an object that processes the contents of a work package.

Functions

- std::string generateUniqueID ()

  Obtain a unique ID for the current process.
- void printStatus (const std::string &message)

  Print a status message to stdout.
- void logEntry (IO::Logsheet &logsheet)

  Send the current log stream to the log device as a debug entry.
- void logMessage (IO::Logsheet &logsheet, const std::string &message)

  Send a log message to the given Logsheet as a debug entry.
- std::shared_ptr
  < BiometricEvaluation::IO::Logsheet > openLogsheet (const std::string &url, const std::string &description)

  Open a Logsheet object for a component of the MPI framework.

Variables

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

G.13.1 Detailed Description

Common declarations and functions for the MPI-based functionality.

G.13.2 Function Documentation

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

Obtain a unique ID for the current process.

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

Returns

The unique ID for the process.

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

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

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

Parameters

in	*logsheet*	The open Logsheet to write into.

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

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

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

Parameters

in	*logsheet*	The open Logsheet to write into.
in	*message*	The log message.

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

Open a Logsheet object for a component of the MPI framework.

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

Parameters

in	*url*	The Uniform Resource Locator for the Logsheet.
in	*description*	The description of the Logsheet.

Returns

Shared pointer to the Logsheet object.

Exceptions

Error::ParameterError	Invalid URL.
Error::Exception	Failed to create the Logsheet object. The exception string will contain more information.

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

Print a status message to stdout.

Parameters

in	*message*	The messasge to be printed.

G.14 BiometricEvaluation::Process Namespace Reference

Process information and controls.

Classes

- class CommandCenter
- class CommandParser
- class ForkManager

  Manager implementation that starts Workers by calling fork(2).
- class ForkWorkerController

  Wrapper of a Worker returned from a Process::ForkManager.
- class Manager

  An interface for intranode process management classes.
- class MessageCenter
- class MessageCenterListener
- class MessageCenterReceiver

  Receives message from a client, forwarding to the central MessageCenter.
- class POSIXThreadManager

  Manager implementation that starts Workers in POSIX threads.
- class POSIXThreadWorkerController

  Decorated Worker returned from a Process::POSIXThreadManager.
- class Semaphore

  Represent a semaphore that can be used for interprocess communication.
- class Statistics

  The Statistics class provides an interface for gathering process statistics, such as memory usage, system time, etc.
- class Worker

  An abstraction of an instance that performs work on given data.
- class WorkerController

  Wrapper of a Worker returned from a Process::Manager.

Typedefs

- using ParameterList = std::map< std::string, std::shared_ptr< void >>

G.14.1 Detailed Description

Process information and controls.

The Process package gathers all process related matters, including a class to obtain resource usage statistics.

G.14.2 Typedef Documentation

using BiometricEvaluation::Process::ParameterList = typedef std::map<std::string, std::shared_ptr<void>>

Convenience alias for parameter lists to child routines

G.15 BiometricEvaluation::System Namespace Reference

Operating system, hardware, etc.

Functions

- uint32_t getCPUCount ()

  Obtain the number of central processing units that are online. Typically, this is the total CPU core count for the system.
- uint64_t getRealMemorySize ()

  Obtain the amount of real memory in the system.
- double getLoadAverage ()

  Obtain the system load average for the last minute.

G.15.1 Detailed Description

Operating system, hardware, etc.

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

G.15.2 Function Documentation

uint32_t BiometricEvaluation::System::getCPUCount ( )

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

Returns

The number of processing units.

Exceptions

Error::NotImplemented	Not implemented for this operating system, or the underlying OS feature is not installed.

double BiometricEvaluation::System::getLoadAverage ( )

Obtain the system load average for the last minute.

Returns

The system load average.

Exceptions

Error::NotImplemented	Not implemented for this operating system, or the underlying OS feature is not installed.

uint64_t BiometricEvaluation::System::getRealMemorySize ( )

Obtain the amount of real memory in the system.

Returns

The real memory size, in kilobytes.

Exceptions

Error::NotImplemented	Not implemented for this operating system, or the underlying OS feature is not installed.

G.16 BiometricEvaluation::Text Namespace Reference

Text processing for string objects.

Functions

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

G.16.1 Detailed Description

Text processing for string objects.

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

G.16.2 Function Documentation

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

Extract the filename component of a pathname.

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

Parameters

in	*path*	Path from which to extract the filename portion.

Returns

Filename portion of path.

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

Compare two ASCII-encoded strings.

Parameters

str1	First string to compare.
str2	Second string to compare.

Returns

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

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

Compute the digest of a string.

Parameters

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

Exceptions

Error::MemoryError	Could not allocate memory to store digest.
Error::NotImplemented	The value of digest is not a supported digest.
Error::StrategyError	An error occurred while obtaining the digest.

Returns

An ASCII representation of the hex digits composing the digest.

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

Compute the digest of a memory buffer.

Parameters

in	*buffer*	The buffer of which a digest should be computed.
in	*buffer_size*	The size of buffer.

in	*digest*	The digest to use. Any digest supported by OpenSSL is valid, and the default is MD5.

Exceptions

Error::MemoryError	Could not allocate memory to store digest.
Error::NotImplemented	The value of digest is not a supported digest.
Error::StrategyError	An error occurred while obtaining the digest.

Returns

An ASCII representation of the hex digits composing the digest.

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

Extract the directory component of a pathname.
Returns the string up to, but not including, the final '/'.

Parameters

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

Returns

Directory portion of path.

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

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

Parameters

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

Returns

Vector of string tokens, in order of appearance.

Note

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

G.17 BiometricEvaluation::Time Namespace Reference

Support for time and timers.

Classes

- class Timer

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

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

Functions

- std::string getCurrentTime ()
- std::string getCurrentDate ()
- std::string getCurrentDateAndTime ()
- std::string getCurrentCalendarInformation (const std::string &formatString)

  Obtain customized calendar information.
- std::string put_time (const struct tm *tmb, const char *fmt)

  Manual implementation of std::put_time.
- std::ostream & operator<< (std::ostream &s, const Timer &timer)

  Output stream operator overload for Timer.
- void **WatchdogSignalHandler** (int signo, siginfo_t *info, void *uap)

Variables

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

G.17.1 Detailed Description

Support for time and timers.

The Time package gathers all timing relating matters, such as Timers, Watchdog timers, etc. Time values are in microsecond units.

G.17.2 Function Documentation

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

Obtain customized calendar information.

Parameters

formatString	A C++11 put_time-compatible format string.

Returns

The current calendar information formatted as specified in formatString.

Note

Return value is undefined if format string is invalid.

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

Returns

The current ISO 8601 date as a string.

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

Returns

The standard locale current date and time as a string.

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

Returns

The current ISO 8601 time as a string.

std::ostream& BiometricEvaluation::Time::operator<< (std::ostream & *s*, const Timer & *timer*)

Output stream operator overload for Timer.

Parameters

s	Stream to append.
timer	Timer whose elapsed time in microseconds should be appended to s.

Returns

s with value of elapsedStr() appended.

Exceptions

BE::Error::StrategyError	Propagated from elapsedStr().

std::string BiometricEvaluation::Time::put_time (const struct tm * *tmb*, const char * *fmt*)

Manual implementation of std::put_time.

Note

Exists because g++ does not currently implement put_time (http://gcc.gnu.org/bugzilla/show_bug.cgi?id=54354)

G.18 BiometricEvaluation::Video Namespace Reference

Basic information relating to video and streams.

Classes

- class Container

  Representation of a video container.
- struct Frame
- class Stream

Enumerations

- enum CodingFormat { **None** = 0, **MPEG1** = 1, **MPEG2** = 2, **MPEG4** = 3, **H264** = 4 }
- enum ContainerFormat { **MPEG1PS** = 1, **MPEG2TS** = 2, **MPEG4PS** = 3, **AVI** = 4 }

G.18.1 Detailed Description

Basic information relating to video and streams.

Common representation of a video stream. Stream objects can only be obtained from Container objects.

The Video package gathers all video related matters, including classes to represent a video stream and video containers.

G.18.2 Enumeration Type Documentation

enum BiometricEvaluation::Video::CodingFormat [strong]

Video coding formats.

enum BiometricEvaluation::Video::ContainerFormat [strong]

Container formats

G.19 BiometricEvaluation::View Namespace Reference

View information.

Classes

- class AN2KView

  A class to represent single biometric view and derived information.
- class AN2KViewVariableResolution

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

  A class to represent single biometric element view.

Functions

- std::ostream & operator<< (std::ostream &stream, const AN2KView::DeviceMonitoringMode &kind)

  Output stream overload for DeviceMonitoringMode.
- std::ostream & operator<< (std::ostream &stream, const AN2KViewVariableResolution::AN2KQuality↩ Metric &qm)

  Output stream overload for AN2KQualityMetric.

G.19.1 Detailed Description

View information.

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

G.19.2 Function Documentation

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

Output stream overload for AN2KQualityMetric.

Parameters

in	*stream*	Stream on which to append formatted AN2KQualityMetric information.
in	*qm*	AN2KQualityMetric information to append to stream.

Returns

stream with a qm textual representation appended.

Appendix H

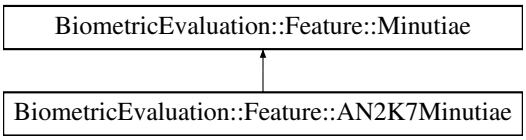
Class Documentation

H.1 BiometricEvaluation::Feature::AN2K7Minutiae Class Reference

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

`#include <be_feature_an2k7minutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::AN2K7Minutiae:



Classes

- struct FingerprintReadingSystem

  Representation of information about a fingerprint reader system.
- class PatternClassification

  Pattern classification codes.

Public Types

- enum EncodingMethod { EncodingMethod::Automatic = 0, EncodingMethod::AutomaticUnedited, EncodingMethod::AutomaticEdited, **Manual** }

  Methods for encoding minutiae data in an AN2K record.
- using **PatternClassificationSet** = std::vector< PatternClassification::Entry >
- using **FingerprintReadingSystem** = struct FingerprintReadingSystem

Public Member Functions

- AN2K7Minutiae (const std::string &filename, int recordNumber)

  Construct an AN2K7 Minutiae object from file data.
- AN2K7Minutiae (Memory::uint8Array &buf, int recordNumber)

  Construct an AN2K7 Minutiae object from data contained in a memory buffer.
- PatternClassificationSet getPatternClassificationSet () const

Obtain the set fingerprint pattern classifications.

- FingerprintReadingSystem getOriginatingFingerprintReadingSystem () const
- MinutiaeFormat getFormat () const

  Obtain the minutiae format kind.
- MinutiaPointSet getMinutiaPoints () const

  Obtain the set of finger minutiae data points. The set may be empty.
- RidgeCountItemSet getRidgeCountItems () const

  Obtain the set of ridge count data items. The set may be empty.
- CorePointSet getCores () const

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

  Obtains the set of delta positions. The set may be empty.

Static Public Member Functions

- static Finger::PatternClassification convertPatternClassification (const char *fpc)

  Convert string read from AN2K record into a PatternClassification.
- static Finger::PatternClassification convertPatternClassification (const PatternClassification::Entry &entry)

  Convert a standard PatternClassification::Entry to a PatternClassification::Kind.
- static EncodingMethod convertEncodingMethod (const char *mem)

  Convert string read from AN2K record into a EncodingMethod.
- static Image::Coordinate convertCoordinate (const char *str, bool calculateDistance=true)

  Obtain a Coordinate given an AN2K entry.

H.1.1 Detailed Description

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

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

H.1.2 Member Enumeration Documentation

enum BiometricEvaluation::Feature::AN2K7Minutiae::EncodingMethod [strong]

Methods for encoding minutiae data in an AN2K record.

Enumerator

Automatic No possible human interaction

AutomaticUnedited Editing possible, but not performed

AutomaticEdited Editing possible and was performed

H.1.3 Constructor & Destructor Documentation

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

Construct an AN2K7 Minutiae object from file data.

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

Parameters

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

Exceptions

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

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

Construct an AN2K7 Minutiae object from data contained in a memory buffer.

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

Parameters

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

Exceptions

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

H.1.4 Member Function Documentation

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

Obtain a Coordinate given an AN2K entry.

This AN2K entry is formatted as "XXXXYYYY".

Parameters

in	*str*	Coordinate string from an AN2K record.
in	*calculate↩ Distance*	Whether or not to calculate the [xy]Distance portion of the Coordinate.

Returns

Image::Coordinate representation of str.

Exceptions

Error::DataError	Invalid format of str.

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

Convert string read from AN2K record into a EncodingMethod.

Parameters

in	*mem*	Value for minutiae encoding method read from AN2K record.

Exceptions

Error::DataError	Invalid value for mem.

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

Convert string read from AN2K record into a PatternClassification.
Parameters

in	*fpc*	Value for pattern classification read from AN2K record.

Exceptions

Error::DataError	Invalid value for fpc.

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

Convert a standard PatternClassification::Entry to a PatternClassification::Kind.
Parameters

in	*entry*	A standard pattern classification entry

Exceptions

Error::DataError	Non-standard pattern classification entry.

FingerprintReadingSystem BiometricEvaluation::Feature::AN2K7Minutiae::getOriginating↩ FingerprintReadingSystem ( ) const

Obtain the originating fingerprint reading system.
Exceptions

Error::ObjectDoesNot↩ Exist	The optional OFR field has been excluded.

PatternClassificationSet BiometricEvaluation::Feature::AN2K7Minutiae::getPatternClassificationSet ( ) const

Obtain the set fingerprint pattern classifications.

The code returned may be a standard code or user-defined. Applications should call isPatternClassification↩ Standard() to check.

H.2 BiometricEvaluation::Finger::AN2KMinutiaeDataRecord Class Reference

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

`#include <be_finger_an2kminutiae_data_record.h>`

Public Member Functions

- AN2KMinutiaeDataRecord (const std::string &filename, int recordNumber)

  Construct an AN2KMinutiaeDataRecord object from data contained in a file on disk.
- AN2KMinutiaeDataRecord (Memory::uint8Array &buf, int recordNumber)

  Construct an AN2KMinutiaeDataRecord object from data contained in a memory buffer.
- std::shared_ptr < Feature::AN2K7Minutiae > getAN2K7Minutiae () const

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

  Return impression type field from Type-9 Record.
- std::map< uint16_t, Memory::uint8Array > getRegisteredVendorBlock (Feature::MinutiaeFormat vendor) const

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

H.2.1 Detailed Description

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

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

H.2.2 Constructor & Destructor Documentation

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

Construct an AN2KMinutiaeDataRecord object from data contained in a file on disk.

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

Parameters

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

Exceptions

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

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

Construct an AN2KMinutiaeDataRecord object from data contained in a memory buffer.

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

Parameters

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

Exceptions

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

H.2.3 Member Function Documentation

std::shared_ptr<Feature::AN2K7Minutiae> BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getAN2K7Minutiae ( ) const

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

Returns

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

Impression BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getImpressionType ( ) const

Return impression type field from Type-9 Record.

Returns

Impression type of the image from which minutiae points were generated.

std::map<uint16_t, Memory::uint8Array> BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getRegisteredVendorBlock (Feature::MinutiaeFormat *vendor*) const

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

Parameters

in	*vendor*	The vendor whose registered minutiae blocks are being requested.

Returns

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

Exceptions

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

H.3 BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric Struct Reference

A structure to represent an AN2K quality metric.

`#include <be_view_an2kview_varres.h>`

Public Attributes

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

H.3.1 Detailed Description

A structure to represent an AN2K quality metric.

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

H.4 BiometricEvaluation::DataInterchange::AN2KRecord Class Reference

A class to represent an entire ANSI/NIST record.

`#include <be_data_interchange_an2k.h>`

Classes

- struct CharacterSet
- struct DomainName

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

Public Types

- using DomainName = struct DomainName
- using CharacterSet = struct CharacterSet

Public Member Functions

- AN2KRecord (const std::string filename)

  Constructor taking an AN2K record from a file.
- AN2KRecord (Memory::uint8Array &buf)

  Constructor taking an AN2K record from a buffer.
- std::string getVersionNumber () const
- std::string getDate () const
- std::string getDestinationAgency () const
- std::string getOriginatingAgency () const
- std::string getTransactionControlNumber () const
- std::string getNativeScanningResolution () const
- std::string getNominalTransmittingResolution () const
- uint32_t getFingerLatentCount () const

  Obtain the count of latent (Type-13) finger views.
- std::vector < Finger::AN2KViewLatent > getFingerLatents () const

  Obtain all latent (Type-13) finger views.
- uint32_t getFingerCaptureCount () const

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

- std::vector
  < Finger::AN2KViewCapture > getFingerCaptures () const

  Obtain all capture (Type-14) finger views.
- std::vector
  < Finger::AN2KMinutiaeDataRecord > getMinutiaeDataRecordSet () const

  Obtain all minutiae (Type-9) data.
- uint8_t getPriority () const

  Obtain the urgency with which a response is required.
- DomainName getDomainName () const

  Obtain the idntifier of the domain name for the user-defined Type-2 logical record implementation.
- struct tm getGreenwichMeanTime () const

  Obain the date and time of encoding in terms of GMT units.
- std::vector< CharacterSet > getDirectoryOfCharacterSets () const

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

Static Public Member Functions

- static std::set< int > recordLocations (Memory::uint8Array &buf, const View::AN2KView::RecordType recordType)

  Find the position within a buffer of all Records of a particular type.
- static std::set< int > recordLocations (const ANSI_NIST *an2k, const View::AN2KView::RecordType recordType)

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

H.4.1 Detailed Description

A class to represent an entire ANSI/NIST record.

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

H.4.2 Member Typedef Documentation

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

Convenience alias for struct CharacterSet

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

Convenience alias for struct DomainName

H.4.3 Constructor & Destructor Documentation

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

Constructor taking an AN2K record from a file.

Parameters

in	*filename*	The name of the file containing the complete ANSI/NIST record.

Exceptions

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

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

Constructor taking an AN2K record from a buffer.

Parameters

in	*buf*	The memory buffer containing the complete ANSI/NIST record.

Exceptions

Error::DataError	An error occurred when processing the AN2K record.

H.4.4 Member Function Documentation

std::string BiometricEvaluation::DataInterchange::AN2KRecord::getDate ( ) const

Returns

The date field in the Type-1 record.

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

Returns

The destination agency ID.

std::vector<CharacterSet> BiometricEvaluation::DataInterchange::AN2KRecord::getDirectoryOf↩ CharacterSets ( ) const

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

Returns

Vector of CharacterSet structs representing other character sets that may appear in the transaction.

DomainName BiometricEvaluation::DataInterchange::AN2KRecord::getDomainName ( ) const

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

Returns

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

uint32_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerCaptureCount ( ) const

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

Returns

The number of captures in the AN2K record.

std::vector<Finger::AN2KViewCapture> BiometricEvaluation::DataInterchange::AN2KRecord::getFingerCaptures ( ) const

Obtain all capture (Type-14) finger views.
The returned vector will be empty when no capture views are present in the AN2KRecord.

Returns

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

uint32_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerLatentCount ( ) const

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

Returns

The number of latents in the AN2K record.

std::vector<Finger::AN2KViewLatent> BiometricEvaluation::DataInterchange::AN2KRecord::getFingerLatents ( ) const

Obtain all latent (Type-13) finger views.
The returned vector will be empty when no latent views are present in the AN2KRecord.

Returns

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

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

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

Returns

struct tm encoding of the GMT field.

std::vector<Finger::AN2KMinutiaeDataRecord> BiometricEvaluation::DataInterchange::AN2KRecord::getMinutiaeDataRecordSet ( ) const

Obtain all minutiae (Type-9) data.

Returns

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

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

Returns

The native scanning resolution.

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

Returns

The nominal transmitting resolution.

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

Returns

The originating agency ID.

uint8_t BiometricEvaluation::DataInterchange::AN2KRecord::getPriority ( ) const

Obtain the urgency with which a response is required.

Returns

Priority (1:High - 9:Low)

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

Returns

The transcantion control number.

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

Returns

The record version field in the Type-1 record.

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

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

Parameters

in	*buf*	AN2K Buffer to search.
in	*recordType*	The ID of the Record to search for.

Returns

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

Exceptions

Error::DataError	An error occurred when processing the AN2K record.

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

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

Parameters

in	*an2k*	ANSI_NIST struct to search.
in	*recordType*	The ID of the Record to search for.

Returns

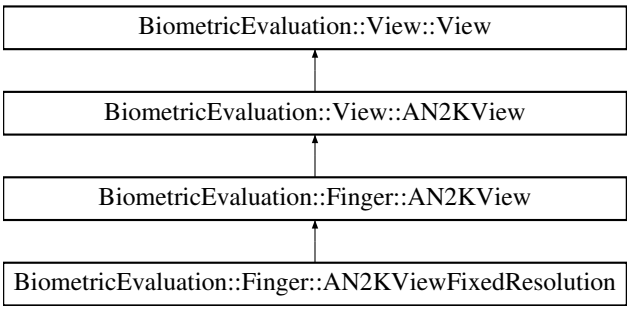
Set of integer positions within the ANSI_NIST struct where a recordType Record is located.

H.5 BiometricEvaluation::Finger::AN2KView Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_an2kview.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KView:



Public Member Functions

- std::vector < AN2KMinutiaeDataRecord > getMinutiaeDataRecordSet () const

  Obtain the set of minutiae records.
- Finger::PositionSet getPositions () const

  Obtain the set of finger positions.
- Finger::Impression getImpressionType () const

  Obtain the finger impression code.

Static Public Member Functions

- static Finger::Position convertPosition (int an2kFGP)

  Convert a compression algorithm indicator from an AN2K finger image record.
- static Finger::PositionSet populateFGP (FIELD *field)

  Read the finger positions from an AN2K record.

- static Finger::Impression convertImpression (const unsigned char *str)

  Convert an impression code from a string.
- static Finger::FingerImageCode convertFingerImageCode (const char *str)

  Convert an finger image code from a string.

Protected Member Functions

- AN2KView (const std::string filename, const RecordType typeID, const uint32_t recordNumber)

  Construct an AN2K finger view from a file.
- AN2KView (Memory::uint8Array &buf, const RecordType typeID, const uint32_t recordNumber)

  Construct an AN2K finger view from a buffer.
- void addMinutiaeDataRecord (Finger::AN2KMinutiaeDataRecord &mdr)

  Add a minutiae data record to the AN2KMinutiaeDataRecord set.
- void setPositions (Finger::PositionSet &ps)

  Add a position set to the collection of position sets.
- void setImpressionType (Finger::Impression &imp)

  Mutator for the impression type.

Additional Inherited Members

H.5.1 Detailed Description

A class to represent single finger view and derived information.

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

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

H.5.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KView::AN2KView (const std::string *filename*, const RecordType *typeID*, const uint32_t *recordNumber*) [protected]

Construct an AN2K finger view from a file.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.

Error::DataError	An error occurred when parsing the AN2K record.
Error::FileError	An error occurred when reading the file.

BiometricEvaluation::Finger::AN2KView::AN2KView (Memory::uint8Array & *buf*, const RecordType *typeID*, const uint32_t *recordNumber*) [protected]

Construct an AN2K finger view from a buffer.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.

H.5.3 Member Function Documentation

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

Add a minutiae data record to the AN2KMinutiaeDataRecord set.

Parameters

in	*mdr*	The minutiae data record to be added.

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

Convert an finger image code from a string.

Parameters

in	*str*	The character string containing the image code.

Returns

A FingerImageCode value.

Exceptions

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

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

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

Parameters

in	*an2kFGP*	A finger position code as defined by the AN2K standard.

Exceptions

Error::DataError	The position code is invalid.

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

Obtain the finger impression code.

Returns

The finger impression code.

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

Obtain the set of minutiae records.

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

Returns

The vector of minutiae data records.

Finger::PositionSet BiometricEvaluation::Finger::AN2KView::getPositions ( ) const

Obtain the set of finger positions.

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

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

Read the finger positions from an AN2K record.

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

Exceptions

Error::DataError	The data contains an invalid value.

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

Mutator for the impression type.

Parameters

in	*imp*	The impression type for this finger view.

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

Add a position set to the collection of position sets.

Parameters

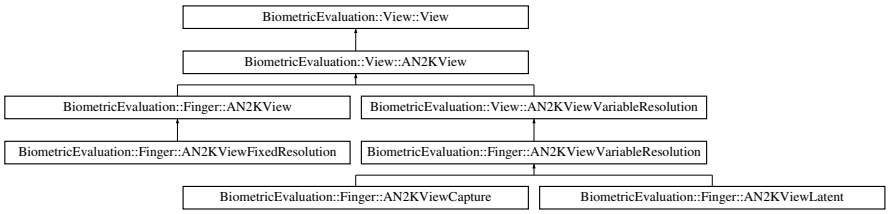
in	*ps*	The position set to be added.

H.6 BiometricEvaluation::View::AN2KView Class Reference

A class to represent single biometric view and derived information.

`#include <be_view_an2kview.h>`

Inheritance diagram for BiometricEvaluation::View::AN2KView:



Public Types

- enum RecordType : uint16_t {
  Type_1 = 1, **Type_2** = 2, **Type_3** = 3, **Type_4** = 4,
  Type_5 = 5, **Type_6** = 6, **Type_7** = 7, **Type_8** = 8,
  Type_9 = 9, **Type_10** = 10, **Type_11** = 11, **Type_12** = 12,
  Type_13 = 13, **Type_14** = 14, **Type_15** = 15, **Type_16** = 16,
  Type_17 = 17, **Type_99** = 99 }
- enum DeviceMonitoringMode {
  DeviceMonitoringMode::Controlled, DeviceMonitoringMode::Assisted, DeviceMonitoringMode::Observed,
  DeviceMonitoringMode::Unattended,
  DeviceMonitoringMode::Unknown, DeviceMonitoringMode::NA }

  The level of human monitoring for the image capture device.

Public Member Functions

- AN2KView (const std::string filename, const RecordType typeID, const uint32_t recordNumber)

  Construct an AN2K view from a file.
- AN2KView (Memory::uint8Array &buf, const RecordType typeID, const uint32_t recordNumber)

  Construct an AN2K view from a buffer.
- std::vector
  < Finger::AN2KMinutiaeDataRecord > getMinutiaeDataRecordSet () const

  Obtain the set of minutiae records.

- RecordType getRecordType () const

  Obtain the ANSI-NIST record type.

Static Public Member Functions

- static DeviceMonitoringMode convertDeviceMonitoringMode (const char *dmm)

  Convert a device monitoring mode indicator from an AN2K record.
- static Image::CompressionAlgorithm convertCompressionAlgorithm (const uint16_t recordType, const unsigned char *an2kValue)

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

Static Public Attributes

- static const double MinimumScanResolutionPPMM

  Constants to define the minimum resolution used for fingerprint images in an AN2k record.
- static const double **HalfMinimumScanResolutionPPMM**
- static const int FixedResolutionBitDepth = 8

  The defined bit-depth for fixed-resolution images.

Protected Member Functions

- Memory::AutoBuffer< ANSI_NIST > getAN2K () const

  Obtain the complete ANSI/NIST record set.
- RECORD * getAN2KRecord () const

  Obtain a pointer to the single ANSI/NIST record.

H.6.1 Detailed Description

A class to represent single biometric view and derived information.

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

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

H.6.2 Member Enumeration Documentation

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

The level of human monitoring for the image capture device.

Enumerator

Controlled Operator physically controls the subject to acquire biometric sample.

Assisted Person available to provide assistance to the subject submitting the biometric.

Observed Person present to observe the operation of the device but provides no assistance.

Unattended No one present to observe or provide assistance.

Unknown No information is known.

NA Optional field – not specified

enum BiometricEvaluation::View::AN2KView::RecordType : uint16_t [strong]

The type of AN2K record.

H.6.3 Constructor & Destructor Documentation

BiometricEvaluation::View::AN2KView::AN2KView (const std::string *filename,* const RecordType *typeID,* const uint32_t *recordNumber*)

Construct an AN2K view from a file.

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

BiometricEvaluation::View::AN2KView::AN2KView (Memory::uint8Array & *buf,* const RecordType *typeID,* const uint32_t *recordNumber*)

Construct an AN2K view from a buffer.

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

H.6.4 Member Function Documentation

static Image::CompressionAlgorithm BiometricEvaluation::View::AN2KView::convert↩ CompressionAlgorithm (const uint16_t *recordType,* const unsigned char * *an2kValue*) [static]

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

Parameters

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

Returns

The compression algorithm.

Exceptions

Error::DataError	Invalid compression algorithm for record type.
Error::ParameterError	Invalid record type.

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

Convert a device monitoring mode indicator from an AN2K record.

Parameters

dmm	Item value for device monitoring mode from an AN2K record.

Returns

DeviceMonitoringMode representation of dmm.

Exceptions

Error::DataError	Invalid format of dmm.

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

Obtain a pointer to the single ANSI/NIST record.

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

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

Obtain the set of minutiae records.

Each AN2KViewVariableResolution may have more than one associated Type-9 record and each Type-9 record may have more than one minutiae format.

Returns

A vector of minutiae data records.

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

Obtain the ANSI-NIST record type.

Returns

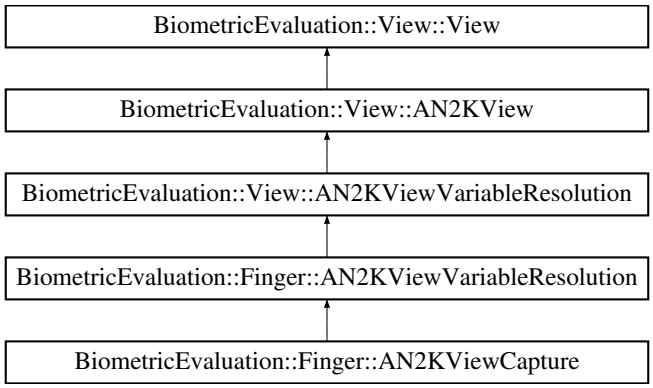
The type of record used to construct this object.

H.7 BiometricEvaluation::Finger::AN2KViewCapture Class Reference

Represents an ANSI/NIST variable-resolution finger image.

`#include <be_finger_an2kview_capture.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewCapture:



Classes

- struct FingerSegmentPosition

  Locations of an individual finger segment in a slap.

Public Types

- enum AmputatedBandaged { AmputatedBandaged::Amputated, AmputatedBandaged::Bandaged, Amputated↩ Bandaged::NA }

  Enumeration of the finger amuptated or bandaged code, a reason that a capture could not be made.
- using **FingerSegmentPosition** = struct FingerSegmentPosition
- using **FingerSegmentPositionSet** = std::vector< FingerSegmentPosition >

Public Member Functions

- AN2KViewCapture (const std::string &filename, const uint32_t recordNumber)

  Construct an AN2K finger view from a file.
- AN2KViewCapture (Memory::uint8Array &buf, const uint32_t recordNumber)

  Construct an AN2K finger view using from a memory buffer.
- QualityMetricSet extractNISTQuality (const FIELD *field)

  Extract the NQM information from an AN2K FIELD.
- PositionDescriptors getPrintPositionDescriptors () const

  Return search position descriptors.
- QualityMetricSet getNISTQualityMetric () const

  Obtain the NIST quality metric for all segmented finger images.
- QualityMetricSet getSegmentationQualityMetric () const

  Obtain the segmentation quality metric for all segmented finger images.
- AmputatedBandaged getAmputatedBandaged () const
- FingerSegmentPositionSet getFingerSegmentPositionSet () const
- FingerSegmentPositionSet getAlternateFingerSegmentPositionSet () const
- QualityMetricSet getFingerprintQualityMetric () const

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

Static Public Member Functions

- static AmputatedBandaged convertAmputatedBandaged (const char *ampcd)

  Convert string read from AN2K record into a AmputatedBandaged code.
- static FingerSegmentPosition convertFingerSegmentPosition (const SUBFIELD *sf)

  Convert SUBFIELD read from AN2K record into a FingerSegmentPosition struct.
- static FingerSegmentPosition convertAlternateFingerSegmentPosition (const SUBFIELD *sf)

  Convert SUBFIELD read from AN2K record into an AlternateFingerSegmentPosition struct.

Additional Inherited Members

H.7.1 Detailed Description

Represents an ANSI/NIST variable-resolution finger image.

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

H.7.2 Member Enumeration Documentation

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

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

Enumerator

Amputated Amputation

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

NA Optional field – not specified

H.7.3 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KViewCapture::AN2KViewCapture (const std::string & *filename,* const uint32_t *recordNumber*)

Construct an AN2K finger view from a file.

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

Parameters

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

Exceptions

Error::ParameterError	
Error::DataError	
Error::FileError	An error occurred when opening or reading the file.

BiometricEvaluation::Finger::AN2KViewCapture::AN2KViewCapture (Memory::uint8Array & *buf,* const uint32_t *recordNumber*)

Construct an AN2K finger view using from a memory buffer.

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

H.7.4 Member Function Documentation

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

Convert SUBFIELD read from AN2K record into an AlternateFingerSegmentPosition struct.

Parameters

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

Exceptions

Error::DataError	Invalid value with sf.

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

Convert string read from AN2K record into a AmputatedBandaged code.

Parameters

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

Exceptions

Error::DataError	Invalid value for ampcd.

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

Convert SUBFIELD read from AN2K record into a FingerSegmentPosition struct.

Parameters

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

Exceptions

Error::DataError	Invalid value within sf.

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

Extract the NQM information from an AN2K FIELD.

Parameters

field	FIELD containing properly formatted NQM data

Returns

QualityMetricSet representation of field.

Exceptions

Error::DataError	Invalid format of field for NQM.

FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getAlternateFinger↩ SegmentPositionSet ( ) const

Returns

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

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

Returns

Optional amputated or bandaged code.

QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::getFingerprintQualityMetric ( ) const

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

Returns

Fingerprint quality metrics

FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getFingerSegment↩ PositionSet ( ) const

Returns

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

QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::getNISTQualityMetric ( ) const

Obtain the NIST quality metric for all segmented finger images.

Returns

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

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

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

Obtain the segmentation quality metric for all segmented finger images.

Returns

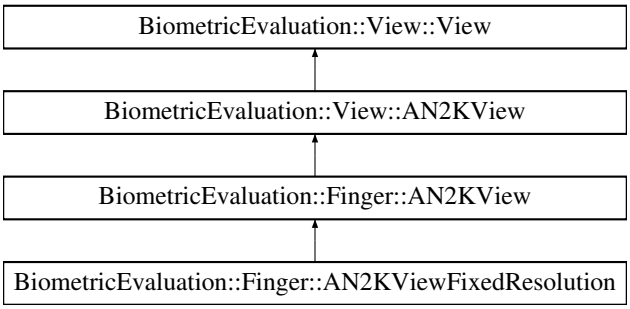
QualityMetricSet containing the segmentation quality metric for all segmented finger images.

H.8 BiometricEvaluation::Finger::AN2KViewFixedResolution Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_an2kview_fixedres.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewFixedResolution:



Public Member Functions

- AN2KViewFixedResolution (const std::string filename, const RecordType typeID, const uint32_t record↩ Number)

  Construct an AN2K finger view from a file.
- AN2KViewFixedResolution (Memory::uint8Array &buf, const RecordType typeID, const uint32_t record↩ Number)

  Construct an AN2K finger view from a buffer.

Additional Inherited Members

H.8.1 Detailed Description

A class to represent single finger view and derived information.

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

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

H.8.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution (const std::string *filename*, const RecordType *typeID*, const uint32_t *recordNumber*)

Construct an AN2K finger view from a file.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.
Error::FileError	An error occurred when reading the file.

BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution (Memory::uint8Array & *buf*, const RecordType *typeID*, const uint32_t *recordNumber*)

Construct an AN2K finger view from a buffer.

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

Parameters

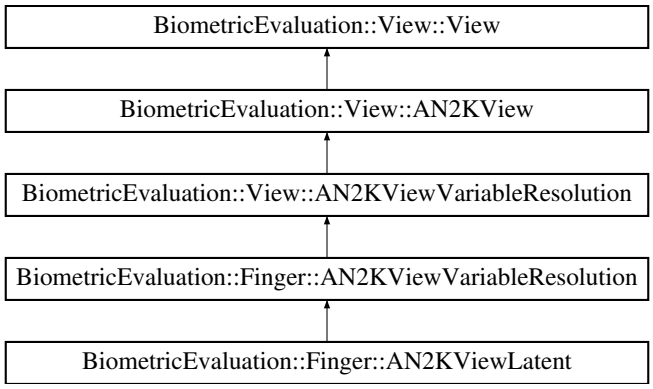
in	*buf*	The buffer containing the AN2K record.
in	*typeID*	The type of AN2K finger view: Type-3/Type-4/etc.
in	*recordNumber*	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.

H.9 BiometricEvaluation::Finger::AN2KViewLatent Class Reference

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewLatent:



Public Member Functions

- AN2KViewLatent (const std::string &filename, const uint32_t recordNumber)

  Construct an AN2K finger view from a file.
- AN2KViewLatent (Memory::uint8Array &buf, const uint32_t recordNumber)

  Construct an AN2K finger view using from a memory buffer.
- QualityMetricSet getLatentQualityMetric () const

  Obtain metrics for latent image quality score data for the image stored in this record.
- PositionDescriptors getSearchPositionDescriptors () const

  Return search position descriptors.

Additional Inherited Members

H.9.1 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KViewLatent::AN2KViewLatent (const std::string & *filename,* const uint32_t *recordNumber*)

Construct an AN2K finger view from a file.
The file must contain the entire AN2K record, not just the finger image and/or minutiae records.

BiometricEvaluation::Finger::AN2KViewLatent::AN2KViewLatent (Memory::uint8Array & *buf,* const uint32_t *recordNumber*)

Construct an AN2K finger view using from a memory buffer.
The buffer must contain the entire AN2K record, not just the finger image and/or minutiae records.

H.9.2 Member Function Documentation

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

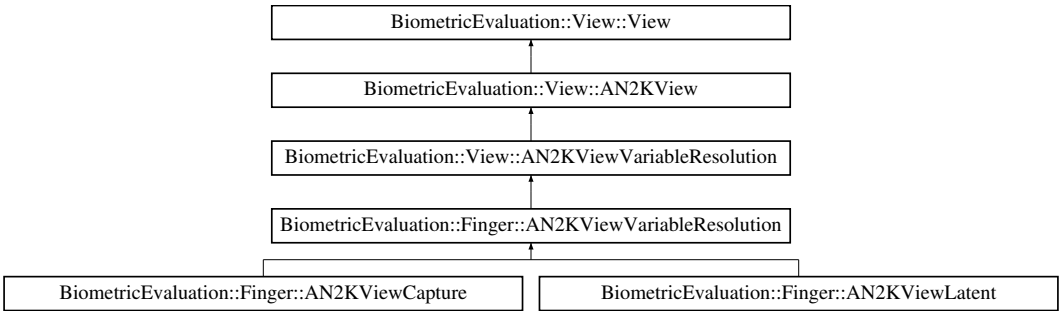
Obtain metrics for latent image quality score data for the image stored in this record.

Returns

Latent quality metrics

H.10 BiometricEvaluation::Finger::AN2KViewVariableResolution Class Reference

A class to represent single finger view based on an ANSI/NIST record.
`#include <be_finger_an2kview_varres.h>`
Inheritance diagram for BiometricEvaluation::Finger::AN2KViewVariableResolution:



Classes

- struct PrintPositionCoordinate

  Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

Public Types

- using **PrintPositionCoordinate** = struct PrintPositionCoordinate
- using **PrintPositionCoordinateSet** = std::vector< PrintPositionCoordinate >

Public Member Functions

- Finger::PositionSet getPositions () const

  Obtain the set of finger positions.
- Finger::Impression getImpressionType () const
- PrintPositionCoordinateSet getPrintPositionCoordinates () const

  Obtain print position coordinates.

Protected Member Functions

- AN2KViewVariableResolution (const std::string &filename, const RecordType typeID, const uint32_t recordNumber)

  Construct an AN2K finger view from a file.
- AN2KViewVariableResolution (Memory::uint8Array &buf, const RecordType typeID, const uint32_t recordNumber)

  Construct an AN2K finger view from a buffer.
- PositionDescriptors getPositionDescriptors () const

Static Protected Member Functions

- static PrintPositionCoordinate convertPrintPositionCoordinate (SUBFIELD *subfield)

  Convert a print position coordinate AN2K subfield to a PrintPositionCoordinate object.
- static PositionDescriptors parsePositionDescriptors (const RecordType typeID, const RECORD *record)

  Parse position descriptors from a record.

Additional Inherited Members

H.10.1 Detailed Description

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

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

H.10.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KViewVariableResolution::AN2KViewVariableResolution (const std::string & *filename*, const RecordType *typeID*, const uint32_t *recordNumber*) [protected]

Construct an AN2K finger view from a file.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.
Error::FileError	An error occurred when reading the file.

BiometricEvaluation::Finger::AN2KViewVariableResolution::AN2KViewVariableResolution (Memory::uint8Array & *buf*, const RecordType *typeID*, const uint32_t *recordNumber*) [protected]

Construct an AN2K finger view from a buffer.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.

H.10.3 Member Function Documentation

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

Convert a print position coordinate AN2K subfield to a PrintPositionCoordinate object.

Parameters

in	*subfield*	A print position coordinate AN2K subfield

Returns

Object representation of field.

Exceptions

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

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

Returns

The finger impression code.

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

Returns

The set of position descriptors.

Finger::PositionSet BiometricEvaluation::Finger::AN2KViewVariableResolution::getPositions ( ) const

Obtain the set of finger positions.

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

PrintPositionCoordinateSet BiometricEvaluation::Finger::AN2KViewVariableResolution::getPrint↩ PositionCoordinates ( ) const

Obtain print position coordinates.

Returns

Set of all PrintPositionCoordinates

static PositionDescriptors BiometricEvaluation::Finger::AN2KViewVariableResolution::parse↩ PositionDescriptors (const RecordType *typeID*, const RECORD * *record*) [static], [protected]

Parse position descriptors from a record.

Parameters

in	*typeID*	The logical record type.
in	*record*	The opened AN2K record.

Returns

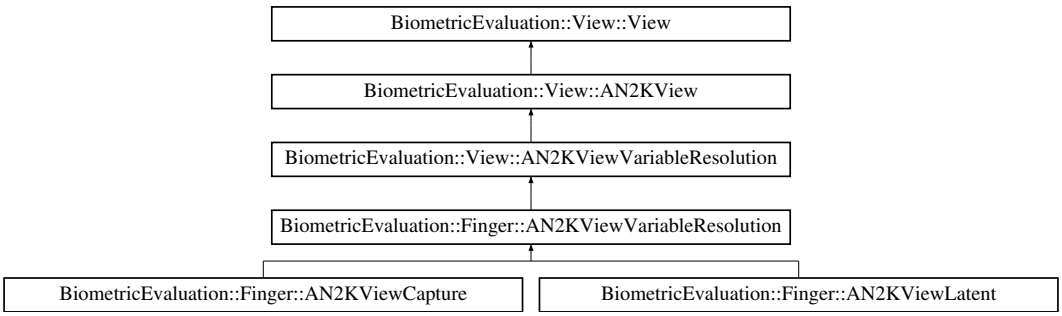
Mapping of finger position codes to finger image code.

H.11 BiometricEvaluation::View::AN2KViewVariableResolution Class Reference

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

`#include <be_view_an2kview_varres.h>`

Inheritance diagram for BiometricEvaluation::View::AN2KViewVariableResolution:



Classes

- struct AN2KQualityMetric

  A structure to represent an AN2K quality metric.

Public Types

- using **AN2KQualityMetric** = struct AN2KQualityMetric
- using **QualityMetricSet** = std::vector< AN2KQualityMetric >

Public Member Functions

- std::string getSourceAgency () const
- std::string getCaptureDate () const
- std::string getComment () const

  Obtain the comment field.
- Memory::uint8Array getUserDefinedField (const uint16_t field) const

  Obtain a user-defined field.

Static Public Member Functions

- static QualityMetricSet extractQuality (FIELD *field)

  Read a Quality Metric Set from a variable resolution AN2K record.
- static Memory::uint8Array parseUserDefinedField (const RECORD *const record, int fieldID)

  Read raw bytes from a user-defined AN2K field.

Protected Member Functions

- AN2KViewVariableResolution (const std::string &filename, const RecordType typeID, const uint32_t recordNumber)

  Construct an AN2K finger view from a file.
- AN2KViewVariableResolution (Memory::uint8Array &buf, const RecordType typeID, const uint32_t recordNumber)

  Construct an AN2K finger view using from a memory buffer.
- QualityMetricSet getQualityMetric () const

  Obtain quality metrics for associated image record.

Additional Inherited Members

H.11.1 Detailed Description

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

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

H.11.2 Constructor & Destructor Documentation

BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution (const std::string & *filename*, const RecordType *typeID*, const uint32_t *recordNumber*) [protected]

Construct an AN2K finger view from a file.

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

BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution (Memory::uint8Array & *buf*, const RecordType *typeID*, const uint32_t *recordNumber*) [protected]

Construct an AN2K finger view using from a memory buffer.

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

H.11.3 Member Function Documentation

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

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

Parameters

in	*field*	A pointer to the field within the AN2K record.

Exceptions

Error::DataError	The data contains an invalid value.

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

Returns

The capture date.

std::string BiometricEvaluation::View::AN2KViewVariableResolution::getComment ( ) const

Obtain the comment field.

The comment field is optional in an AN2K record.

Returns

The comment field, empty string if not present.

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

Obtain quality metrics for associated image record.

Returns

Quality metrics

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

Returns

The source agency.

Memory::uint8Array BiometricEvaluation::View::AN2KViewVariableResolution::getUserDefinedField (const uint16_t *field*) const

Obtain a user-defined field.

Fields are retrieved on-demand and then cached.

Parameters

in	*field*	The field number to retrieve.

Returns

Raw bytes read from the field.

Exceptions

Error::ParameterError	Invalid value for field.

static Memory::uint8Array BiometricEvaluation::View::AN2KViewVariableResolution::parseUserDefinedField (const RECORD *const *record*, int *fieldID*) [static]

Read raw bytes from a user-defined AN2K field.

Parameters

in	*record*	Pointer to a RECORD containing the user-defined field.
in	*fieldID*	The user-defined field number.

Returns

Raw bytes from field.

Exceptions

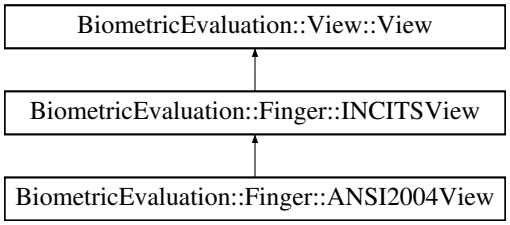
Error::ParameterError	Invalid value for fieldID.

H.12 BiometricEvaluation::Finger::ANSI2004View Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_ansi2004view.h>`

Inheritance diagram for BiometricEvaluation::Finger::ANSI2004View:



Public Member Functions

- ANSI2004View ()

  Construct an empty ANSI finger view.
- ANSI2004View (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber)

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

- ANSI2004View (Memory::uint8Array &fmrBuffer, Memory::uint8Array &firBuffer, const uint32_t viewNumber)

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

Protected Member Functions

- void **readFMRHeader** (Memory::IndexedBuffer &buf)
- void readCoreDeltaData (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)

  Read the core points data.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x20323000

Additional Inherited Members

H.12.1 Detailed Description

A class to represent single finger view and derived information.

A Finger::ANSI2004View object represents a finger view from a INCITS/ANSI-2004 Finger Minutiae Record.

H.12.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::ANSI2004View::ANSI2004View (const std::string & *fmrFilename,* const std::string & *firFilename,* const uint32_t *viewNumber*)

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

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

Parameters

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

BiometricEvaluation::Finger::ANSI2004View::ANSI2004View (Memory::uint8Array & *fmrBuffer,* Memory::uint8Array & *firBuffer,* const uint32_t *viewNumber*)

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

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

Parameters

in	*fmrBuffer*	The buffer containing the complete finger minutiae record.

in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

H.12.3 Member Function Documentation

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

Read the core points data.

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

Parameters

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

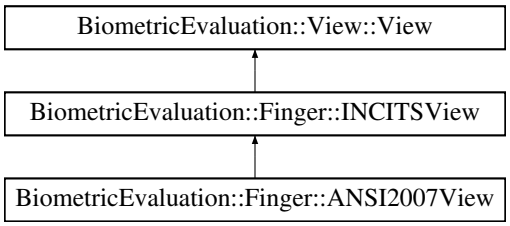
Implements BiometricEvaluation::Finger::INCITSView.

H.13 BiometricEvaluation::Finger::ANSI2007View Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_ansi2007view.h>`

Inheritance diagram for BiometricEvaluation::Finger::ANSI2007View:



Public Member Functions

- ANSI2007View (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber)

  Construct an ANSI-2007 finger view from records contained in files.
- ANSI2007View (Memory::uint8Array &fmrBuffer, Memory::uint8Array &firBuffer, const uint32_t viewNumber)

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

Protected Member Functions

- void **readFMRHeader** (Memory::IndexedBuffer &buf)
- void **readFVMR** (Memory::IndexedBuffer &buf)
- void readCoreDeltaData (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)

  Read the core points data.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x30333000

Additional Inherited Members

H.13.1 Detailed Description

A class to represent single finger view and derived information.

A Finger::ANSI2007View object represents a finger view from a INCITS/ANSI-2007 Finger Minutiae Record.

H.13.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::ANSI2007View::ANSI2007View (const std::string & *fmrFilename,* const std::string & *firFilename,* const uint32_t *viewNumber*)

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

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

Parameters

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

Exceptions

Error::DataError	Invalid record format.

BiometricEvaluation::Finger::ANSI2007View::ANSI2007View (Memory::uint8Array & *fmrBuffer,* Memory::uint8Array & *firBuffer,* const uint32_t *viewNumber*)

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

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

Parameters

in	*fmrBuffer*	The buffer containing the complete finger minutiae record.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.

H.13.3 Member Function Documentation

void BiometricEvaluation::Finger::ANSI2007View::readCoreDeltaData (Memory::IndexedBuffer & *buf,* uint32_t *dataLength,* Feature::CorePointSet & *cores,* Feature::DeltaPointSet & *deltas*) `[protected], [virtual]`

Read the core points data.

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

Parameters

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

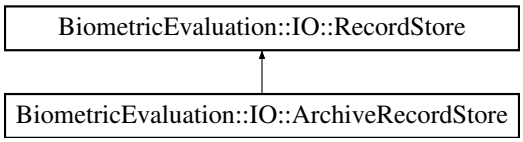
Implements BiometricEvaluation::Finger::INCITSView.

H.14 BiometricEvaluation::IO::ArchiveRecordStore Class Reference

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

`#include <be_io_archiverecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ArchiveRecordStore:



Public Member Functions

- ArchiveRecordStore (const std::string &pathname, const std::string &description)
- ArchiveRecordStore (const std::string &pathname, uint8_t mode=IO::READWRITE)
- ~ArchiveRecordStore ()
- uint64_t getSpaceUsed () const

  Obtain real storage utilization.
- void sync () const
- void insert (const std::string &key, const void *const data, const uint64_t size)
- void remove (const std::string &key)
- uint64_t read (const std::string &key, void *const data) const
- void replace (const std::string &key, const void *const data, const uint64_t size)
- uint64_t length (const std::string &key) const
- void flush (const std::string &key) const
- uint64_t sequence (std::string &key, void *const data=nullptr, int cursor=BE_RECSTORE_SEQ_NEXT)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (const std::string &key)
- void move (const std::string &pathname)

  Move the RecordStore.
- bool needsVacuum ()
- std::string getArchiveName () const
- std::string getManifestName () const
- **ArchiveRecordStore** (const ArchiveRecordStore &)=delete
- ArchiveRecordStore & **operator=** (const ArchiveRecordStore &)=delete

Static Public Member Functions

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

Static Public Attributes

- static const std::string MANIFEST_FILE_NAME
- static const std::string ARCHIVE_FILE_NAME
- static const long OFFSET_RECORD_REMOVED = -1

Additional Inherited Members

H.14.1 Detailed Description

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

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

Each record is assigned a string key, which will be required for retrieving the data. As the data is written, a plain text entry is entered into the manifest in the format:
key offset size
where offset is the offset into the archive file key's data chunk resides and size is the length of key's data chunk.

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

H.14.2 Constructor & Destructor Documentation

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

Create a new ArchiveRecordStore, read/write mode.
Parameters

in	*pathname*	The directory where the store is to be created.
in	*description*	The store's description.

Exceptions

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

BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore (const std::string & *pathname,* uint8_t *mode* = *IO::READWRITE*)

Open an existing ArchiveRecordStore.
Parameters

in	*pathname*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

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

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

Destructor.

H.14.3 Member Function Documentation

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

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

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

Returns

Path to archive file.

std::string BiometricEvaluation::IO::ArchiveRecordStore::getManifestName ( ) const

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

Returns

Path to manifest file.

uint64_t BiometricEvaluation::IO::ArchiveRecordStore::getSpaceUsed ( ) const [virtual]

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ArchiveRecordStore::insert (const std::string & *key*, const void *const *data*, const uint64_t *size*) [virtual]

Insert a record into the store.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::ArchiveRecordStore::length (const std::string & *key*) const [virtual]

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Move the RecordStore.

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

Parameters

in	*pathname*	The new path of the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

bool BiometricEvaluation::IO::ArchiveRecordStore::needsVacuum ( )

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

Returns

true if vacuum() would be beneficial false otherwise

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

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

Parameters

in	*pathname*	The path 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.

Returns

true if vacuum() would be beneficial false otherwise

uint64_t BiometricEvaluation::IO::ArchiveRecordStore::read (const std::string & *key*, void *const *data*) const [virtual]

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

Parameters

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

Returns

The size of the record.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

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

Remove a record from the store.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ArchiveRecordStore::replace (const std::string & *key*, const void *const *data*, const uint64_t *size*) [virtual]

Replace a complete record in a store.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::ArchiveRecordStore::sequence (std::string & *key*, void *const *data* = *nullptr*, int *cursor* = BE_RECSTORE_SEQ_NEXT) [virtual]

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

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

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

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

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

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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

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

Parameters

in	*pathname*	The pathname 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.

Note

This is an expensive operation.

H.14.4 Member Data Documentation

const std::string BiometricEvaluation::IO::ArchiveRecordStore::ARCHIVE_FILE_NAME [static]

Name of the archive file on disk

const std::string BiometricEvaluation::IO::ArchiveRecordStore::MANIFEST_FILE_NAME [static]

Name of the manifest file on disk

const long BiometricEvaluation::IO::ArchiveRecordStore::OFFSET_RECORD_REMOVED = -1 [static]

Offset placeholder indicating a removed record

H.15 BiometricEvaluation::Memory::AutoArray< T > Class Template Reference

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

`#include <be_memory_autoarray.h>`

Public Types

- using value_type = T
- using size_type = size_t
- using iterator = AutoArrayIterator< false, T >
- using const_iterator = AutoArrayIterator< true, T >
- using reference = T &
- using const_reference = const T &

Public Member Functions

- operator T * ()

  Convert AutoArray to T array.
- operator const T * () const

  Convert AutoArray to const T array.
- reference operator[ ] (ptrdiff_t index)

  Subscripting operator overload with unchecked access.
- const_reference operator[ ] (ptrdiff_t index) const

  Const subscripting operator overload with unchecked access.
- reference at (ptrdiff_t index)

  Subscript into the AutoArray with checked access.
- const_reference at (ptrdiff_t index) const

  Subscript into the AutoArray with checked access.
- iterator begin ()

  Obtain an iterator to the beginning of the AutoArray.
- const_iterator begin () const

  Obtain an iterator to the beginning of the AutoArray.
- const_iterator cbegin () const

  Obtain an iterator to the beginning of the AutoArray.
- iterator end ()

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

Obtain an iterator to the end of the AutoArray.

- const_iterator cend () const

  Obtain an iterator to the end of the AutoArray.

- size_type size () const

  Obtain the number of accessible elements.

- void resize (size_type new_size, bool free=false)

  Change the number of accessible elements.

- void copy (const T *buffer)

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

- void copy (const T *buffer, size_type size)

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

- AutoArray (size_type size=0)

  Construct an AutoArray.

- AutoArray (const AutoArray &copy)

  Construct an AutoArray.

- AutoArray (AutoArray &&rvalue) noexcept

  Construct an AutoArray.

- AutoArray & operator= (const AutoArray &other)

  Copy assignment operator overload performing a deep copy.

- AutoArray & operator= (AutoArray &&other) noexcept(noexcept(std::swap(std::declval< value_type & >(), std::declval< value_type & >()))&&noexcept(std::swap(std::declval< size_type & >(), std::declval< size_type & >())))

  Move assignment operator.

- ~AutoArray ()

H.15.1 Detailed Description

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

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

H.15.2 Member Typedef Documentation

template<class T> using BiometricEvaluation::Memory::AutoArray< T >::const_iterator = AutoArrayIterator<true, T>

Const iterator of element

template<class T> using BiometricEvaluation::Memory::AutoArray< T >::const_reference = const T&

Const reference element

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

Iterator of element

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

Reference to element

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

Type of subscripts, counts, etc.

template<class T> using BiometricEvaluation::Memory::AutoArray< T >::value_type = T

Type of element

H.15.3 Constructor & Destructor Documentation

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

Construct an AutoArray.

Parameters

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

Exceptions

Error::MemoryError	Could not allocate new memory.

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.

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.

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

Destructor

H.15.4 Member Function Documentation

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

Subscript into the AutoArray with checked access.

Parameters

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.

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.

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

Obtain an iterator to the beginning of the AutoArray.

Returns

Iterator positioned at the first element of the AutoArray.

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

Obtain an iterator to the beginning of the AutoArray.

Returns

Const iterator positioned at the first element of the AutoArray.

template<class T > BiometricEvaluation::Memory::AutoArray< T >::const_iterator BiometricEvaluation::Memory::AutoArray< T >::cbegin ( ) const

Obtain an iterator to the beginning of the AutoArray.

Returns

Const iterator positioned at the first element of the AutoArray.

template<class T > BiometricEvaluation::Memory::AutoArray< T >::const_iterator BiometricEvaluation::Memory::AutoArray< T >::cend ( ) const

Obtain an iterator to the end of the AutoArray.

Returns

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

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

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

Parameters

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.

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.

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

Obtain an iterator to the end of the AutoArray.

Returns

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

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

Obtain an iterator to the end of the AutoArray.

Returns

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

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

Convert AutoArray to const T array.

Returns

Const pointer to the beginning of the underlying array storage.

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

Convert AutoArray to T array.

Returns

Pointer to the beginning of the underlying array storage.

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

Copy assignment operator overload performing a deep copy.

Parameters

in	*other*	AutoArray to be copied.

Returns

Reference to a new AutoArray object, the lvalue AutoArray.

Exceptions

Error::MemoryError	Could not allocate new memory.

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

Move assignment operator.

Parameters

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

Returns

Reference to the lvalue AutoArray.

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

Subscripting operator overload with unchecked access.

Parameters

in	*index*	Subscript into underlying storage.

Returns

Reference to the element at the specified index.

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

Const subscripting operator overload with unchecked access.

Parameters

in	*index*	Subscript into underlying storage.

Returns

Const reference to the element at the specified index.

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

Change the number of accessible elements.

Parameters

in	*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.

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

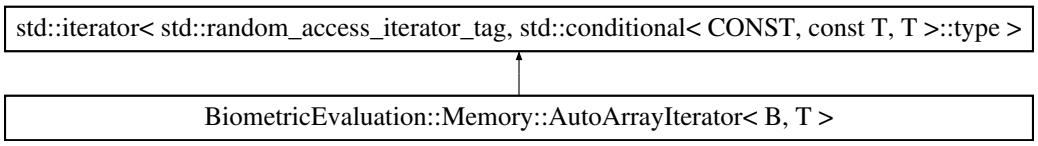
If resize() has been called, the value returned from size() may be smaller than the actual allocated size of the underlying storage.

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

RandomAccessIterator for any AutoArray.

`#include <be_memory_autoarrayiterator.h>`

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



Public Types

- using CONTAINER = typename std::conditional< CONST, const AutoArray< T > *, AutoArray< T > * >::type

  Convenience definition for a reference to the iterated type with appropriate constness.
- using POINTER = typename std::conditional< CONST, const typename AutoArrayIterator< CONST, T >::pointer, typename AutoArrayIterator< CONST, T >::pointer >::type

  Convenience definition for a pointer to the iterated type with appropriate constness.
- using REFERENCE = typename std::conditional< CONST, const typename AutoArrayIterator< CONST, T >::reference, typename AutoArrayIterator< CONST, T >::reference >::type

  Convenience definition for a reference to the iterated type with appropriate constness.
- using DIFFERENCE = typename AutoArrayIterator< CONST, T >::difference_type

Public Member Functions

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

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

Friends

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

H.16.1 Detailed Description

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

RandomAccessIterator for any AutoArray.

Note

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

H.16.2 Member Typedef Documentation

template<bool B, class T > using BiometricEvaluation::Memory::AutoArrayIterator< B, T >::DIFFERENCE = typename AutoArrayIterator<CONST, T>::difference_type

Convenience definition for difference_type

H.16.3 Constructor & Destructor Documentation

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

Default constructor.
Parameters

autoArray	Pointer to the AutoArray to iterate
offset	The offset into the AutoArray where this iterator should start.

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

Default copy constructor

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

Default move constructor

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

Default destructor

H.16.4 Member Function Documentation

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

Returns

Whether or not the offsets are different.

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

Returns

Object at the current offset.

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

Returns

This object with offset incremented by rhs' offset.

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

Returns

This object with offset incremented rhs.

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

Returns

This object with incremented offset.

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

Returns

This object before incrementing offset.

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

Returns

This object with rhs added to offset.

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

Returns

Offset decremented by rhs' offset.

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

Returns

This object with offset decremented rhs.

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

Returns

This object with decremented offset.

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

Returns

This object before decrementing offset.

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

Returns

This object with rhs removed from offset.

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

Returns

Address of object at the current offset.

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

Returns

true if this offset is < rhs'.

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

Returns

true if this offset is <= rhs'.

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

Returns

This object with offset set to rhs.

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

Default assignment operator.

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

Returns

Whether or not the offsets are the same.

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

Returns

true if this offset is > rhs'.

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

Returns

true if this offset is >= rhs'.

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

Returns

Object at rhs.

H.16.5 Friends And Related Function Documentation

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

Returns

New iterator combining offsets.

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

Returns

New iterator differing offsets, iterating rhs' AutoArray.

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

Public Types

- using value_type = T

  Manage a memory buffer.
- using **reference** = T &
- using **const_reference** = const T &

Public Member Functions

- **operator T** * ()
- T * **operator->** ()
- AutoBuffer & **operator=** (const AutoBuffer &other)
- **AutoBuffer** (T *data)
- **AutoBuffer** (int(*ctor)(T **), void(*dtor)(T *), int(*copyCtor)(T **, T *)=nullptr)
- **AutoBuffer** (const AutoBuffer &copy)

H.17.1 Member Typedef Documentation

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

Manage a memory buffer.

It's easier to think of AutoBuffer as a wrapper for a pointer rather than the object it truly is. Therefore, you can interact with the AutoBuffer object exactly how you would a traditional pointer, without worrying about memory management.

Say you wanted to use an ANSI_NIST* but didn't want to be responsible for allocating or freeing the memory. Create an AutoBuffer object like:

```
AutoBuffer<ANSI_NIST> obj = AutoBuffer(allocator_fn,
    deallocator_fn[, copy_constructor]);
```

Notice the AutoBuffer is for ANSI_NIST and not ANSI_NIST*, since AutoBuffer will handle the pointer for you. You can pass the AutoBuffer<ANSI_NIST> object to any function that takes an ANSI_NIST*. For example, it's perfectly valid to pass our 'obj' object above to:

```
write_fmttext(FILE *, ANSI_NIST *)
```

If you want to access a member from 'obj', you can use the dereference operator just like you would on a regular ANSI_NIST*:

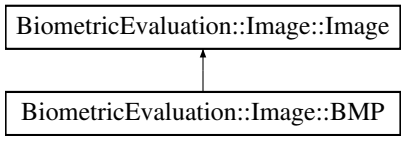
```
int size = obj->num_bytes;
```

H.18 BiometricEvaluation::Image::BMP Class Reference

A BMP-encoded image.

`#include <be_image_bmp.h>`

Inheritance diagram for BiometricEvaluation::Image::BMP:



Public Member Functions

- **BMP** (const uint8_t *data, const uint64_t size)
- Memory::AutoArray< uint8_t > getRawData () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const

  Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isBMP (const uint8_t *data, uint64_t size)

Additional Inherited Members

H.18.1 Detailed Description

A BMP-encoded image.

Note

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

H.18.2 Member Function Documentation

Memory::AutoArray<uint8_t> BiometricEvaluation::Image::BMP::getRawData ( ) const [virtual]

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

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError	Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

Memory::AutoArray<uint8_t> BiometricEvaluation::Image::BMP::getRawGrayscaleData (uint8_t *depth* = 8) const [virtual]

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

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

Note

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

Implements BiometricEvaluation::Image::Image.

static bool BiometricEvaluation::Image::BMP::isBMP (const uint8_t * *data*, uint64_t *size*) [static]

Whether or not data is a BMP image.
Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a BMP image, false otherwise.

H.19 BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet Struct Reference

Public Member Functions

- CharacterSet (uint16_t identifier=0, std::string commonName="", std::string version="")

  Create a new CharacterSet struct.

Public Attributes

- uint16_t identifier
- std::string commonName
- std::string version

H.19.1 Constructor & Destructor Documentation

BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::CharacterSet (uint16_t *identifier* = 0, std::string *commonName* = "", std::string *version* = "") [inline]

Create a new CharacterSet struct.

Parameters

identifier	Numeric identifier of the character set.
commonName	Common name of the character set.
version	Optional version number of the character set.

H.19.2 Member Data Documentation

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

Common name of the character set

uint16_t BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::identifier

Identifier (000-999)

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

Optional version of the character set

H.20 BiometricEvaluation::Process::CommandCenter< T, typename >↩ ::Command Class Reference

`#include <be_process_commandcenter.h>`

Public Attributes

- uint32_t clientID
- T command
- std::vector< std::string > arguments

H.20.1 Detailed Description

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>>class BiometricEvaluation::Process::CommandCenter< T, typename >::Command

Parsed command received from the network.

H.20.2 Member Data Documentation

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> std::vector<std::string> BiometricEvaluation::Process::CommandCenter< T, typename >::Command::arguments

Arguments passed to command (optional).

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> uint32_t BiometricEvaluation::Process::CommandCenter< T, typename >::Command::clientID

ID of the sender.

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> T BiometricEvaluation::Process::CommandCenter< T, typename >::Command::command

Enumeration value of the command.

H.21 BiometricEvaluation::Process::CommandCenter< T, typename > Class Template Reference

`#include <be_process_commandcenter.h>`

Classes

- class Command

Public Member Functions

- CommandCenter (uint16_t port=MessageCenter::DEFAULT_PORT)

  Constructor.
- ~CommandCenter ()=default
- bool hasPendingCommands ()

  Determine if there are commands waiting.
- bool getNextCommand (Command &command, int numSeconds=-1, std::string invalidCommandResponse="")

  Get the next command.
- void sendResponse (uint32_t clientID, const std::string &response, const std::string prefix=">> ", const std::string suffix="\n")

  Send a string response to a client.
- void disconnectClient (uint32_t clientID)

  Break the connection with a client.

H.21.1 Detailed Description

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>>class BiometricEvaluation::Process::CommandCenter< T, typename >

Receive enumerations as commands over the network.

H.21.2 Constructor & Destructor Documentation

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> BiometricEvaluation::Process::CommandCenter< T, typename >::CommandCenter (uint16_t *port* = MessageCenter::DEFAULT_PORT) [inline]

Constructor.
Parameters

port	Port to listen on for commands.

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> BiometricEvaluation::Process::CommandCenter< T, typename >::~CommandCenter ( ) [default]

Destructor (default).

H.21.3 Member Function Documentation

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> void BiometricEvaluation::Process::CommandCenter< T, typename >::disconnectClient (uint32_t *clientID*) [inline]

Break the connection with a client.
Parameters

clientID	ID of the client to disconect.

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> bool BiometricEvaluation::Process::CommandCenter< T, typename >::getNextCommand (Command & *command*, int *numSeconds* = -1, std::string *invalidCommandResponse* = "") [inline]

Get the next command.
Parameters

command	Reference to a Command that will be populated when this method returns true.
numSeconds	Number of seconds to wait for a command, or -1 to block indefinitely.
invalid↩ Command↩ Response	Optional string to send, such as usage, that will be sent when an unrecognized command is received.

Returns

true if command has been populated, false otherwise.

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> bool BiometricEvaluation::Process::CommandCenter< T, typename >::hasPendingCommands ( ) [inline]

Determine if there are commands waiting.

Returns

true if there are commands waiting, false otherwise.

Note

Returns immediately.

See also

BiometricEvaluation::Process::CommandCenter:: getNextCommand()

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>> void BiometricEvaluation::Process::CommandCenter< T, typename >::sendResponse (uint32_t *clientID*, const std::string & *response*, const std::string *prefix* = ">> ", const std::string *suffix* = "\n") [inline]

Send a string response to a client.

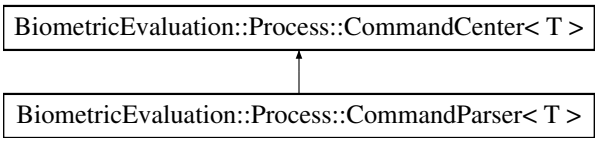
Parameters

clientID	ID of client to communicate with.
response	Printable string to send to client.
prefix	String to prefix to responses.
suffix	String to append to responses.

H.22 BiometricEvaluation::Process::CommandParser< T > Class Template Reference

`#include <be_process_commandcenter.h>`

Inheritance diagram for BiometricEvaluation::Process::CommandParser< T >:



Public Member Functions

- virtual void parse (const typename CommandCenter< T >::Command &command)=0

  Parse command.
- bool getNextCommand (typename CommandCenter< T >::Command &command, int numSeconds=-1)

  Get the next command.
- void setUsage (const std::string &usage)

  String sent when an invalid command is received.
- std::string getUsage () const
- CommandParser (uint16_t port=MessageCenter::DEFAULT_PORT)

  Constructor.
- virtual ~CommandParser ()=default

H.22.1 Detailed Description

template<typename T>class BiometricEvaluation::Process::CommandParser< T >

Abstraction to parse messages received via CommandCenter.

H.22.2 Constructor & Destructor Documentation

template<typename T > BiometricEvaluation::Process::CommandParser< T >::CommandParser (uint16_t *port* = MessageCenter::DEFAULT_PORT) `[inline]`

Constructor.

Parameters

port	Port to listen on for commands.

template<typename T > virtual BiometricEvaluation::Process::CommandParser< T >::~CommandParser ( ) [virtual], [default]

Virtual destructor (default).

H.22.3 Member Function Documentation

template<typename T > bool BiometricEvaluation::Process::CommandParser< T >::getNextCommand (typename CommandCenter< T >::Command & *command,* int *numSeconds* = *-1*) [inline]

Get the next command.
Parameters

command	Reference to a Command that will be populated when this method returns true.
numSeconds	Number of seconds to wait for a command, or -1 to block indefinitely.

Returns

true if command has been populated, false otherwise.

template<typename T > std::string BiometricEvaluation::Process::CommandParser< T >::getUsage ( ) const [inline]

Returns

Usage string.

template<typename T > virtual void BiometricEvaluation::Process::CommandParser< T >::parse (const typename CommandCenter< T >::Command & *command*) [pure virtual]

Parse command.
Implement this method as a switch statement of your command enumeration.

template<typename T > void BiometricEvaluation::Process::CommandParser< T >::setUsage (const std::string & *usage*) [inline]

String sent when an invalid command is received.
Parameters

usage	String to send when an invalid command is received.

Note

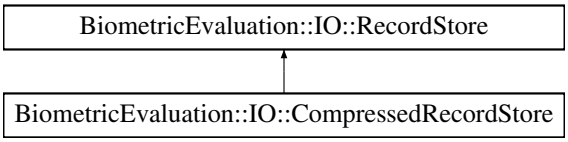
If not set, no additional usage is sent.

H.23 BiometricEvaluation::IO::CompressedRecordStore Class Reference

Sibling-implemented RecordStore with Compression.

`#include <be_io_compressedrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::CompressedRecordStore:



Public Member Functions

- CompressedRecordStore (const std::string &pathname, const std::string &description, const RecordStore::Kind &recordStoreType, const std::string &compressorType)
- CompressedRecordStore (const std::string &pathname, const std::string &description, const RecordStore::Kind &recordStoreType, const Compressor::Kind &compressorType)
- CompressedRecordStore (const std::string &pathname, uint8_t mode=IO::READWRITE)
- uint64_t getSpaceUsed () const

  Obtain real storage utilization.
- void sync () const
- void insert (const std::string &key, const void *const data, const uint64_t size)
- void remove (const std::string &key)
- uint64_t read (const std::string &key, void *const data) const
- void replace (const std::string &key, const void *const data, const uint64_t size)
- uint64_t length (const std::string &key) const
- void flush (const std::string &key) const
- uint64_t sequence (std::string &key, void *const data=nullptr, int cursor=BE_RECSTORE_SEQ_NEXT)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (const std::string &key)
- void move (const std::string &pathname)

  Move the RecordStore.
- CompressedRecordStore (const CompressedRecordStore &rhs)=delete

  Copy constructor (disabled).
- CompressedRecordStore & operator= (const CompressedRecordStore &rhs)=delete

  Assignment operator (disabled).

Static Public Attributes

- static const std::string BACKING_STORE
- static const std::string COMPRESSOR_TYPE_KEY

Additional Inherited Members

H.23.1 Detailed Description

Sibling-implemented RecordStore with Compression.

H.23.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const std::string & *pathname,* const std::string & *description,* const RecordStore::Kind & *recordStoreType,* const std::string & *compressorType*)

Create a new CompressedRecordStore, read/write mode.

Parameters

in	*pathname*	The directory where the store is to be created.
in	*description*	The store's description.
in	*recordStoreType*	The type of RecordStore subclass the internal RecordStores should be.
in	*compressorType*	The type of compression that should be used within the internal RecordStores.

Exceptions

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

BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const std::string & *pathname*, const std::string & *description*, const RecordStore::Kind & *recordStoreType*, const Compressor::Kind & *compressorType*)

Create a new CompressedRecordStore, read/write mode.

Parameters

in	*pathname*	The directory where the store is to be created.
in	*description*	The store's description.
in	*recordStoreType*	The type of RecordStore subclass the internal RecordStores should be.
in	*compressorType*	The type of compression that should be used within the internal RecordStores.

Exceptions

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

BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const std::string & *pathname*, uint8_t *mode* = `IO::READWRITE`)

Open an existing CompressedRecordStore.

Parameters

in	*pathname*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

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

BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const CompressedRecordStore & *rhs*) `[delete]`

Copy constructor (disabled).

Disabled because this object could represent a file on disk.

Parameters

rhs	CompressedRecordStore object to copy.

H.23.3 Member Function Documentation

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

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::CompressedRecordStore::getSpaceUsed ( ) const [virtual]

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::CompressedRecordStore::insert (const std::string & *key*, const void *const *const *data*, const uint64_t *size*) [virtual]

Insert a record into the store.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::CompressedRecordStore::length (const std::string & *key*) const [virtual]

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Move the RecordStore.

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

Parameters

in	*pathname*	The new path of the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

CompressedRecordStore& BiometricEvaluation::IO::CompressedRecordStore::operator= (const CompressedRecordStore & *rhs*) [delete]

Assignment operator (disabled).

Disabled because this object could represent a file on disk.

Parameters

rhs	CompressedRecordStore object to assign.

Returns

CompressedRecordStore object, now containing the contents of rhs.

uint64_t BiometricEvaluation::IO::CompressedRecordStore::read (const std::string & *key*, void *const *const *data*) const [virtual]

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

Parameters

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

Returns

The size of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Remove a record from the store.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::CompressedRecordStore::replace (const std::string & *key*, const void *const *data*, const uint64_t *size*) [virtual]

Replace a complete record in a store.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::CompressedRecordStore::sequence (std::string & *key*, void *const *data* = *nullptr*, int *cursor* = BE_RECSTORE_SEQ_NEXT) [virtual]

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

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

out	*key*	The key of the currently sequenced record.

in	*data*	Pointer to where the data is to be written. Applications can set data to nullptr to indicate only the key is wanted.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::CompressedRecordStore::setCursorAtKey (const std::string & *key*) [virtual]

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

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::CompressedRecordStore::sync ( ) const [virtual]

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

H.23.4 Member Data Documentation

const std::string BiometricEvaluation::IO::CompressedRecordStore::BACKING_STORE [static]

Name of the underlying store within this RS

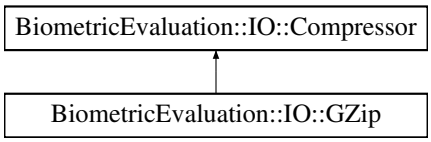
const std::string BiometricEvaluation::IO::CompressedRecordStore::COMPRESSOR_TYPE_KEY [static]

Name of the key storing compressor type

H.24 BiometricEvaluation::IO::Compressor Class Reference

`#include <be_io_compressor.h>`

Inheritance diagram for BiometricEvaluation::IO::Compressor:



Public Types

- enum Kind { **GZIP** }

Public Member Functions

- Compressor ()

  Create a new Compressor object.
- virtual Memory::uint8Array compress (const uint8_t *const uncompressedData, uint64_t uncompressed↩ DataSize) const =0

  Compress a buffer.
- virtual Memory::uint8Array compress (const Memory::uint8Array &uncompressedData) const =0

  Compress a buffer.
- virtual void compress (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize, const std::string &outputFile) const =0

  Compress a buffer.
- virtual void compress (const Memory::uint8Array &uncompressedData, const std::string &outputFile) const =0

  Compress a buffer.
- virtual Memory::uint8Array compress (const std::string &inputFile) const =0

  Compress a file.
- virtual void compress (const std::string &inputFile, const std::string &outputFile) const =0

  Compress a file.
- virtual Memory::uint8Array decompress (const uint8_t *const compressedData, uint64_t compressed↩ DataSize) const =0

  Decompress a compressed buffer.
- virtual Memory::uint8Array decompress (const Memory::uint8Array &compressedData) const =0

  Decompress a compressed buffer.
- virtual Memory::uint8Array decompress (const std::string &inputFile) const =0

  Decompress a compressed buffer into a file.
- virtual void decompress (const Memory::uint8Array &compressedData, const std::string &outputFile) const =0

  Decompress a file.
- virtual void decompress (const uint8_t *const compressedData, const uint64_t compressedDataSize, const std::string &outputFile) const =0

  Decompress a file.

- virtual void decompress (const std::string &inputFile, const std::string &outputFile) const =0

  Decompress a file.
- void setOption (const std::string &optionName, const std::string &optionValue)

  Assign a compressor option.
- void setOption (const std::string &optionName, int64_t optionValue)

  Assign a compressor option.
- std::string getOption (const std::string &optionName) const

  Obtain a compressor option as an integer.
- int64_t getOptionAsInteger (const std::string &optionName) const

  Obtain a compressor option as an integer.
- void removeOption (const std::string &optionName)

  Remove a compressor option.
- virtual ~Compressor ()
- Compressor (const Compressor &other)=delete

  Copy constructor (disabled).
- Compressor & operator= (const Compressor &other)=delete

  Assignment overload (disabled).

Static Public Member Functions

- static std::shared_ptr
  < Compressor > createCompressor (Compressor::Kind compressorKind=Kind::GZIP)

H.24.1 Detailed Description

Implementations for compressing and decompressing data

H.24.2 Member Enumeration Documentation

enum BiometricEvaluation::IO::Compressor::Kind `[strong]`

Kinds of Compressors (for factory)

H.24.3 Constructor & Destructor Documentation

BiometricEvaluation::IO::Compressor::Compressor ( )

Create a new Compressor object.
Default compression options will be used.

virtual BiometricEvaluation::IO::Compressor::~Compressor ( ) `[virtual]`

Destructor

BiometricEvaluation::IO::Compressor::Compressor (const Compressor & *other*) `[delete]`

Copy constructor (disabled).
Disabled because Properties member cannot be copied.

Parameters

other	Compressor to copy.

H.24.4 Member Function Documentation

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (const uint8_t *const *uncompressedData*, uint64_t *uncompressedDataSize*) const [pure virtual]

Compress a buffer.
Parameters

uncompressed↩ Data	Uncompressed data buffer to compress.
uncompressed↩ DataSize	Size of uncompressedData.

Returns

Compressed buffer.

Exceptions

Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (const Memory::uint8Array & *uncompressedData*) const [pure virtual]

Compress a buffer.
Parameters

uncompressed↩ Data	Uncompressed data buffer to compress.

Returns

Compressed buffer.

Exceptions

Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::compress (const uint8_t *const *uncompressedData*, uint64_t *uncompressedDataSize*, const std::string & *outputFile*) const [pure virtual]

Compress a buffer.
Parameters

uncompressed↩ Data	Uncompressed data buffer to compress.

uncompressed↩ DataSize	Size of uncompressedData.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::compress (const Memory::uint8Array & *uncompressedData*, const std::string & *outputFile*) const [pure virtual]

Compress a buffer.
Parameters

uncompressed↩ Data	Uncompressed data buffer to compress.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (const std::string & *inputFile*) const [pure virtual]

Compress a file.
Parameters

inputFile	Path to file to compress.

Returns

Compressed buffer.

Exceptions

Error::ObjectDoesNot↩ Exist	Input file does not exist.
Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::compress (const std::string & *inputFile*, const std::string & *outputFile*) const [pure virtual]

Compress a file.
Parameters

inputFile	Path to file to compress.
outputFile	Path to location where compressed version will be saved.

Exceptions

Error::ObjectDoesNotExist	Input file does not exist.
Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

static std::shared_ptr<Compressor> BiometricEvaluation::IO::Compressor::createCompressor (Compressor::Kind *compressorKind* = `Kind::GZIP`) `[static]`

Compressor factory.

Parameters

compressorKind	A known kind of compressor.

Returns

A new compressor with default options.

Exceptions

Error::ObjectDoesNotExist	Invalid compressor type.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (const uint8_t *const *const *compressedData*, uint64_t *compressedDataSize*) const `[pure virtual]`

Decompress a compressed buffer.

Parameters

compressedData	Compressed data buffer to decompress.
compressedDataSize	Size of compressedData.

Returns

Decompressed data.

Exceptions

Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (const Memory::uint8Array & *compressedData*) const `[pure virtual]`

Decompress a compressed buffer.

Parameters

compressed↩ Data	Compressed data buffer to decompress.

Returns

Decompressed data.

Exceptions

Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (const std::string & *inputFile*) const [pure virtual]

Decompress a compressed buffer into a file.

Parameters

inputFile	Location to save compressed file.

Returns

Decompressed data.

Exceptions

Error::StrategyError	Error in decompression unit.
Error::ObjectDoesNot↩ Exists	Output file already exists.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::decompress (const Memory::uint8Array & *compressedData*, const std::string & *outputFile*) const [pure virtual]

Decompress a file.

Parameters

compressed↩ Data	Compressed data buffer to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::decompress (const uint8_t *const *compressedData*, const uint64_t *compressedDataSize*, const std::string & *outputFile*) const [pure virtual]

Decompress a file.

Parameters

compressedData	Compressed data buffer to decompress.
compressedDataSize	Size of compressedData.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::decompress (const std::string & *inputFile*, const std::string & *outputFile*) const [pure virtual]

Decompress a file.
Parameters

inputFile	Path to file to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectDoesNotExist	Input file does not exist.
Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

std::string BiometricEvaluation::IO::Compressor::getOption (const std::string & *optionName*) const

Obtain a compressor option as an integer.
Parameters

optionName	Name of the option to obtain.

Returns

Value of compressor option.

int64_t BiometricEvaluation::IO::Compressor::getOptionAsInteger (const std::string & *optionName*) const

Obtain a compressor option as an integer.
Parameters

optionName	Name of the option to obtain.

Returns

Value of compressor option.

Exceptions

Error::ObjectDoesNot↩ Exist	The option was never set.

Compressor& BiometricEvaluation::IO::Compressor::operator= (const Compressor & *other*) [delete]

Assignment overload (disabled).

Disabled because Properties member cannot be assigned.

Parameters

other	Compressor to assign.

Returns

lhs Compressor.

void BiometricEvaluation::IO::Compressor::removeOption (const std::string & *optionName*)

Remove a compressor option.

Parameters

optionName	Name of the option to remove.

void BiometricEvaluation::IO::Compressor::setOption (const std::string & *optionName*, const std::string & *optionValue*)

Assign a compressor option.

Will overwrite existing values without warning.

Parameters

optionName	Name of the option to add.
optionValue	Value of the option.

Exceptions

Error::StrategyError	Error setting option.

void BiometricEvaluation::IO::Compressor::setOption (const std::string & *optionName*, int64_t *optionValue*)

Assign a compressor option.

Will overwrite existing values without warning.

Parameters

optionName	Name of the option to add.
optionValue	Value of the option.

Exceptions

Error::StrategyError	Error setting option.

H.25 BiometricEvaluation::Framework::ConstEnumMapWrapper< T > Class Template Reference

Wrapper class around an individual enumeration entity (const).

`#include <be_framework_enumeration.h>`

Public Member Functions

- ConstEnumMapWrapper (const T &enumeration)
- operator std::string () const
- operator T () const

H.25.1 Detailed Description

template<typename T>class BiometricEvaluation::Framework::ConstEnumMapWrapper< T >

Wrapper class around an individual enumeration entity (const).

Because the operators are in the main namespace for maximum usefulness, we must create this additional type to avoid type ambiguity when using more than one template (e.g., string) in a source file.

H.25.2 Constructor & Destructor Documentation

template<typename T > BiometricEvaluation::Framework::ConstEnumMapWrapper< T >::ConstEnumMapWrapper (const T & *enumeration*)

Constructor

H.25.3 Member Function Documentation

template<typename T > BiometricEvaluation::Framework::ConstEnumMapWrapper< T >::operator std::string ( ) const

Implicit conversion to std::string

template<typename T > BiometricEvaluation::Framework::ConstEnumMapWrapper< T >::operator T ( ) const

Implicit conversion to enumeration

H.26 BiometricEvaluation::Video::Container Class Reference

Representation of a video container.

`#include <be_video_container.h>`

Public Member Functions

- Container (const Memory::uint8Array &buffer)

  Construct a Container from a memory buffer.
- Container (const std::shared_ptr< Memory::uint8Array > &buffer)

  Construct a Container from a memory buffer wrapped in a shared pointer.
- Container (const std::string &filename)

  Construct a Container from file.
- uint32_t getAudioCount ()

  Obtain the number of audio streams.
- uint32_t getVideoCount ()

  Obtain the number of video streams.
- std::unique_ptr< Video::Stream > getVideoStream (uint32_t videoNum)

  Obtain a video stream from the container. Video streams are indexed independently from other streams in the container.

H.26.1 Detailed Description

Representation of a video container.

The Container class represents a single container stream that can be used to access the video and audio components of the stream.

H.26.2 Constructor & Destructor Documentation

BiometricEvaluation::Video::Container::Container (const Memory::uint8Array & *buffer*)

Construct a Container from a memory buffer.

Using this constructor can result in buffer memory usage twice that of other constructors.

Exceptions

Error::MemoryError	Error allocating memory for internal buffering.
Error::StrategyError	Other error when reading the container stream.

BiometricEvaluation::Video::Container::Container (const std::shared_ptr< Memory::uint8Array > & *buffer*)

Construct a Container from a memory buffer wrapped in a shared pointer.

Applications must not modify the data underlying the AutoArray.

Exceptions

Error::MemoryError	Error allocating memory for internal buffering.
Error::StrategyError	Other error when reading the container stream.

BiometricEvaluation::Video::Container::Container (const std::string & *filename*)

Construct a Container from file.

Exceptions

Error::ObjectDoesNot↩ Exist	File does not exist.
Error::MemoryError	Error allocating memory for internal buffering.
Error::StrategyError	Other error when reading the container stream.

H.26.3 Member Function Documentation

std::unique_ptr<Video::Stream> BiometricEvaluation::Video::Container::getVideoStream (uint32_t *videoNum*)

Obtain a video stream from the container. Video streams are indexed independently from other streams in the container.

Parameters

videoNum	The number of the video stream within the container.

Exceptions

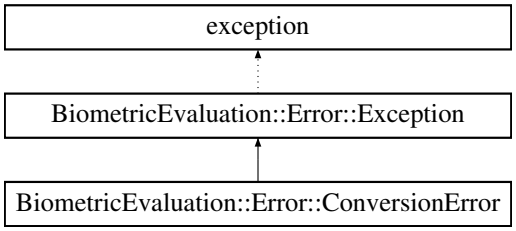
Error::ParameterError	The requested video stream is not available.

H.27 BiometricEvaluation::Error::ConversionError Class Reference

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

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ConversionError:



Public Member Functions

- ConversionError ()
- ConversionError (std::string info)

H.27.1 Detailed Description

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

H.27.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ConversionError::ConversionError ( )

Construct a ConversionError object with the default information string.

BiometricEvaluation::Error::ConversionError::ConversionError (std::string *info*)

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

H.28 BiometricEvaluation::Image::Coordinate Struct Reference

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

`#include <be_image.h>`

Public Member Functions

- Coordinate (const uint32_t x=0, const uint32_t y=0, const float xDistance=0, const float yDistance=0)

  Create a Coordinate struct.

Public Attributes

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

H.28.1 Detailed Description

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

H.28.2 Constructor & Destructor Documentation

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

Create a Coordinate struct.

Parameters

in	*x*	X-coordinate
in	*y*	Y-coordinate
in	*xDistance*	X-coordinate distance from origin
in	*yDistance*	Y-coordinate distance from origin

H.28.3 Member Data Documentation

uint32_t BiometricEvaluation::Image::Coordinate::x

X-coordinate

float BiometricEvaluation::Image::Coordinate::xDistance

X-coordinate distance from origin

uint32_t BiometricEvaluation::Image::Coordinate::y

Y-coordinate

float BiometricEvaluation::Image::Coordinate::yDistance

Y-coordinate distance from origin

H.29 BiometricEvaluation::Feature::CorePoint Struct Reference

Representation of the core.

`#include <be_feature_minutiae.h>`

Public Member Functions

- CorePoint (Image::Coordinate coordinate, bool has_angle=false, int angle=0)

  Create a CorePoint struct.

Public Attributes

- Image::Coordinate **coordinate**
- bool **has_angle**
- int **angle**

H.29.1 Detailed Description

Representation of the core.

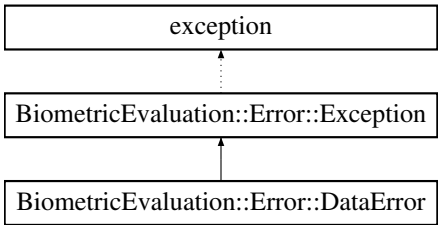
A core has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

H.30 BiometricEvaluation::Error::DataError Class Reference

Error when reading data from an external source.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::DataError:



Public Member Functions

- DataError ()
- DataError (std::string info)

H.30.1 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.

H.30.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::DataError::DataError ( )

Construct a DataError object with the default information string.

BiometricEvaluation::Error::DataError::DataError (std::string *info*)

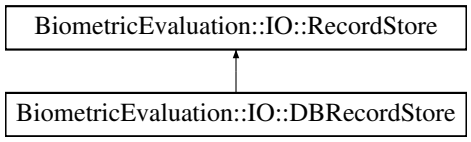
Construct a DataError object with an information string appended to the default information string.

H.31 BiometricEvaluation::IO::DBRecordStore Class Reference

A class that implements IO::RecordStore using a Berkeley DB database as the underlying record storage system.

`#include <be_io_dbrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::DBRecordStore:



Public Member Functions

- DBRecordStore (const std::string &pathname, const std::string &description)
- DBRecordStore (const std::string &pathname, uint8_t mode=IO::READWRITE)
- uint64_t getSpaceUsed () const

  Obtain real storage utilization.
- void sync () const
- void insert (const std::string &key, const void *const data, const uint64_t size)
- void remove (const std::string &key)
- uint64_t read (const std::string &key, void *const data) const
- void replace (const std::string &key, const void *const data, const uint64_t size)
- uint64_t length (const std::string &key) const
- void flush (const std::string &key) const
- uint64_t sequence (std::string &key, void *const data=nullptr, int cursor=BE_RECSTORE_SEQ_NEXT)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (const std::string &key)
- void move (const std::string &pathname)

  Move the RecordStore.
- **DBRecordStore** (const DBRecordStore &)=delete
- DBRecordStore & **operator=** (const DBRecordStore &)=delete

Additional Inherited Members

H.31.1 Detailed Description

A class that implements IO::RecordStore using a Berkeley DB database as the underlying record storage system.

H.31.2 Constructor & Destructor Documentation

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

Create a new DBRecordStore, read/write mode.

Parameters

in	*pathname*	The directory where the store will be created.
in	*description*	The store's description.

Exceptions

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

BiometricEvaluation::IO::DBRecordStore::DBRecordStore (const std::string & *pathname*, uint8_t *mode* = `IO::READWRITE`)

Open an existing DBRecordStore.

Parameters

in	*name*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

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

H.31.3 Member Function Documentation

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

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::DBRecordStore::insert (const std::string & *key*, const void *const *data*, const uint64_t *size*) [virtual]

Insert a record into the store.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::DBRecordStore::length (const std::string & *key*) const [virtual]

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Move the RecordStore.

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

Parameters

in	*pathname*	The new path of the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::DBRecordStore::read (const std::string & *key*, void *const *data*) const [virtual]

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

Parameters

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

Returns

The size of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Remove a record from the store.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::DBRecordStore::replace (const std::string & *key*, const void *const *data*, const uint64_t *size*) [virtual]

Replace a complete record in a store.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::DBRecordStore::sequence (std::string & *key*, void *const *data* = *nullptr*, int *cursor* = BE_RECSTORE_SEQ_NEXT) [virtual]

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

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

Parameters

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

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::DBRecordStore::setCursorAtKey (const std::string & *key*) [virtual]

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

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::DBRecordStore::sync ( ) const [virtual]

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

H.32 BiometricEvaluation::Feature::DeltaPoint Struct Reference

Representation of the delta.

`#include <be_feature_minutiae.h>`

Public Member Functions

- DeltaPoint (Image::Coordinate coordinate, bool has_angle=false, int angle1=0, int angle2=0, int angle3=0)

  Create a DeltaPoint struct.

Public Attributes

- Image::Coordinate **coordinate**
- bool **has_angle**
- int **angle1**
- int **angle2**
- int **angle3**

H.32.1 Detailed Description

Representation of the delta.

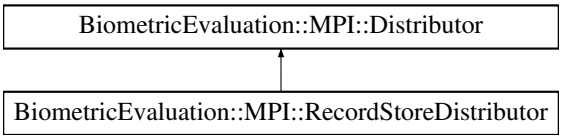
A delta has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

H.33 BiometricEvaluation::MPI::Distributor Class Reference

A class to represent an MPI task that distributes work to other tasks.

`#include <be_mpi_distributor.h>`

Inheritance diagram for BiometricEvaluation::MPI::Distributor:



Public Member Functions

- Distributor (const std::string &propertiesFileName)

  Constructor with properties file name.
- void start ()

  Start of MPI processing for the distributor.

Protected Member Functions

- virtual void createWorkPackage (MPI::WorkPackage &workPackage)=0

  Create a work package for distribution.
- std::shared_ptr< IO::Logsheet > getLogsheet () const

  Get access to the Logsheet object.

H.33.1 Detailed Description

A class to represent an MPI task that distributes work to other tasks.

A Distributor object is based on a set of properties contained in a file. This class must be subclassed and an implementation of the createWorkPackage() method provided.

The distributor sends an MPI message to each receiver object indicating whether it should start and ready for accepting work packages, or proceed immediately to the shutdown state. Failure to start the Distributor object will result in the entire MPI job shutting down before any work is done.

If the Logsheet URL property is set, log messages will be written to that sheet. Otherwise, log messages will be written to a Null Logsheet.

See also

IO::Properties
MPI::Receiver
MPI::WorkPackage

H.33.2 Constructor & Destructor Documentation

BiometricEvaluation::MPI::Distributor::Distributor (const std::string & *propertiesFileName*)

Constructor with properties file name.
Parameters

in	*propertiesFile↩ Name*	The name of the file containing the properties for the new object.

Exceptions

Error::Exception	An error occurred, possibly due to missing or invalid properties.

H.33.3 Member Function Documentation

virtual void BiometricEvaluation::MPI::Distributor::createWorkPackage (MPI::WorkPackage & *workPackage*) [protected], [pure virtual]

Create a work package for distribution.

Implementations of this class create a work package to encapsulate the specific data type that is to be distributed.

Implemented in BiometricEvaluation::MPI::RecordStoreDistributor.

std::shared_ptr<IO::Logsheet> BiometricEvaluation::MPI::Distributor::getLogsheet ( ) const [protected]

Get access to the Logsheet object.

Returns

A shared pointer for the Logsheet object.

void BiometricEvaluation::MPI::Distributor::start ( )

Start of MPI processing for the distributor.

Once started, the distributor will send a message to each receiver task telling it to start and waiting for status back from each receiver.

H.34 BiometricEvaluation::DataInterchange::AN2KRecord::Domain↩ Name Struct Reference

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

`#include <be_data_interchange_an2k.h>`

Public Member Functions

- DomainName (std::string identifier="", std::string version="")

  Create a DomainName struct.

Public Attributes

- std::string identifier
- std::string version

H.34.1 Detailed Description

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

H.34.2 Constructor & Destructor Documentation

BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::DomainName (std::string *identifier* = `""`, std::string *version* = `""`) `[inline]`

Create a DomainName struct.
Parameters

identifier	Unique identifier for agency, entity, or implementation.
version	Optional unique version number of the implementation of the identifier.

H.34.3 Member Data Documentation

std::string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::identifier

Unique identifier for agency, entity, or implementation.

std::string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::version

Optional version of the implementation

H.35 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry Struct Reference

Public Member Functions

- Entry (bool standard, std::string code)

Public Attributes

- bool standard
- std::string code

H.35.1 Constructor & Destructor Documentation

BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::Entry (bool *standard*, std::string *code*)

Create an Entry struct.

Parameters

standard	Whether or not code is a standard AN2K pattern classification code.
code	AN2K or user-defined pattern classification code.

H.35.2 Member Data Documentation

std::string BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::code

AN2K or user-defined pattern classification code.

bool BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::standard

Whether code is a standard AN2K pattern classification code.

H.36 BiometricEvaluation::Framework::EnumerationFunctions< T > Class Template Reference

`#include <be_framework_enumeration.h>`

Static Public Attributes

- static const std::map< T, std::string > enumToStringMap

H.36.1 Detailed Description

template<typename T>class BiometricEvaluation::Framework::EnumerationFunctions< T >

Class to store enumeration/string mappings.

H.36.2 Member Data Documentation

template<typename T > const std::map<T, std::string> BiometricEvaluation::Framework::↩ EnumerationFunctions< T >::enumToStringMap `[static]`

Enumeration -> String Representation

H.37 BiometricEvaluation::Framework::EnumMapWrapper< T > Class Template Reference

Wrapper class around an individual enumeration entity (non-const).

`#include <be_framework_enumeration.h>`

Public Member Functions

- EnumMapWrapper (T &enumeration)
- operator std::string ()
- operator T ()

H.37.1 Detailed Description

template<typename T>class BiometricEvaluation::Framework::EnumMapWrapper< T >

Wrapper class around an individual enumeration entity (non-const).

Because the operators are in the main namespace for maximum usefulness, we must create this additional type to avoid type ambiguity when using more than one template (e.g., string) in a source file.

H.37.2 Constructor & Destructor Documentation

template<typename T > BiometricEvaluation::Framework::EnumMapWrapper< T >::EnumMapWrapper (T & *enumeration*)

Constructor

H.37.3 Member Function Documentation

template<typename T > BiometricEvaluation::Framework::EnumMapWrapper< T >::operator std::string ( )

Implicit conversion to std::string

template<typename T > BiometricEvaluation::Framework::EnumMapWrapper< T >::operator T ( )

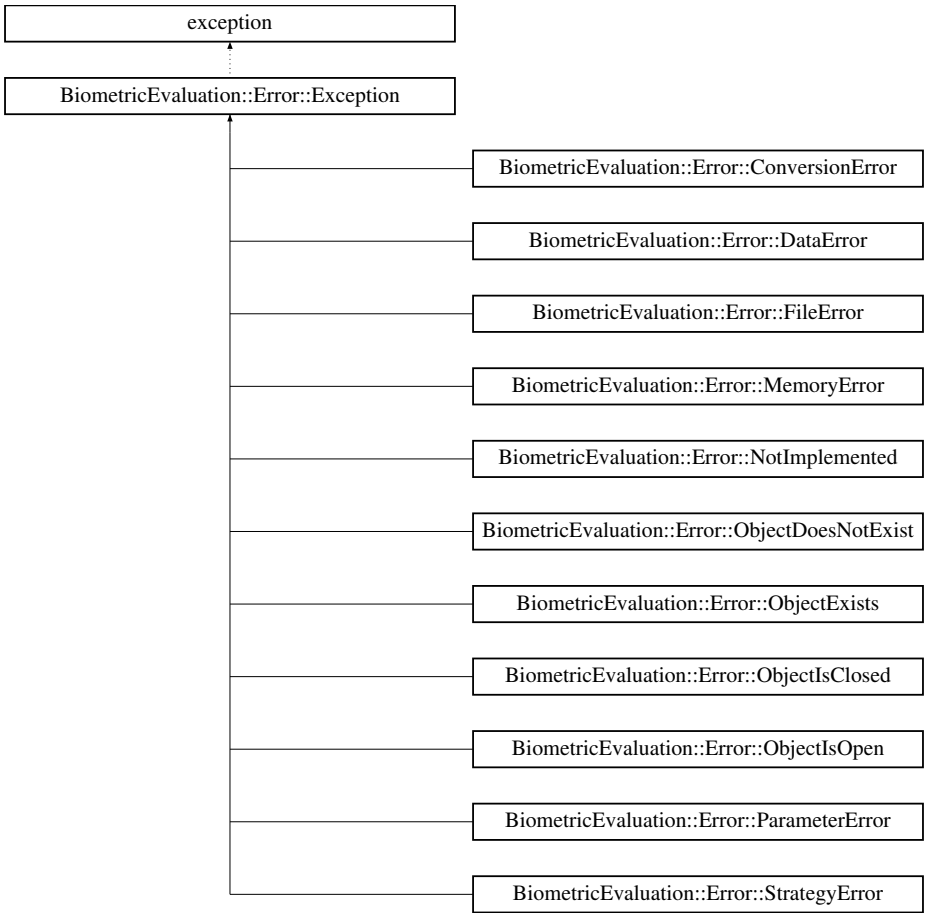
Implicit conversion to enumeration

H.38 BiometricEvaluation::Error::Exception Class Reference

The parent class of all BiometricEvaluation exceptions.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::Exception:



Public Member Functions

- Exception ()
- Exception (std::string info)
- const char * what () const noexcept
- const std::string whatString () const noexcept

H.38.1 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.

H.38.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::Exception::Exception ( )

Construct an Exception object without an information string.

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

Construct an Exception object with an information string.

Parameters

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

H.38.3 Member Function Documentation

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

Obtain the information string associated with the exception.

Returns

The information string as a char array.

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

Obtain the information string associated with the exception.

Returns

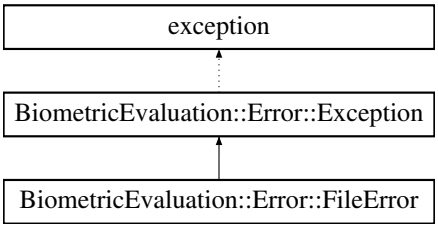
The information string.

H.39 BiometricEvaluation::Error::FileError Class Reference

File error when opening, reading, writing, etc.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::FileError:



Public Member Functions

- FileError ()
- FileError (std::string info)

H.39.1 Detailed Description

File error when opening, reading, writing, etc.

H.39.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::FileError::FileError ( )

Construct a FileError object with the default information string.

BiometricEvaluation::Error::FileError::FileError (std::string *info*)

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

H.40 BiometricEvaluation::IO::FileLogCabinet Class Reference

`#include <be_io_filelogcabinet.h>`

Public Member Functions

- FileLogCabinet (const std::string &pathname, const std::string &description)
- FileLogCabinet (const std::string &pathname)
- std::shared_ptr< FileLogsheet > newLogsheet (const std::string &name, const std::string &description)
- std::string getPathname ()
- std::string getDescription ()
- unsigned int getCount ()

H.40.1 Detailed Description

A class to represent a collection of log sheets.

H.40.2 Constructor & Destructor Documentation

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

Create a new FileLogCabinet in the file system.

Parameters

in	*pathname*	The pathname where the FileLogCabinet is to be created.
in	*description*	The text used to describe the cabinet.

Exceptions

Error::ObjectExists	The cabinet was previously created.
Error::StrategyError	An error occurred when using the underlying file system.

BiometricEvaluation::IO::FileLogCabinet::FileLogCabinet (const std::string & *pathname*)

Open an existing FileLogCabinet.

Parameters

in	*pathname*	The pathname where the FileLogCabinet is located.

Exceptions

Error::ObjectDoesNot↩ Exist	The cabinet does not exist in the file system.

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

H.40.3 Member Function Documentation

unsigned int BiometricEvaluation::IO::FileLogCabinet::getCount ( )

Obtain the number of items in the FileLogCabinet.
@ returns The number of logsheets manages by the cabinet.

std::string BiometricEvaluation::IO::FileLogCabinet::getDescription ( )

Obtain the description of the FileLogCabinet.
@ returns The description of the FileLogCabinet.

std::string BiometricEvaluation::IO::FileLogCabinet::getPathname ( )

Obtain the pathname of the FileLogCabinet.
@ returns The pathname of the FileLogCabinet.

std::shared_ptr<FileLogsheet> BiometricEvaluation::IO::FileLogCabinet::newLogsheet (const std::string & *name*, const std::string & *description*)

Create a new FileLogsheet within the cabinet.
Parameters

in	*name*	The name of the FileLogsheet to be created. This can not be a path name.
in	*description*	The text used to describe the sheet. This text is written into the log file prior to any entries.

Returns

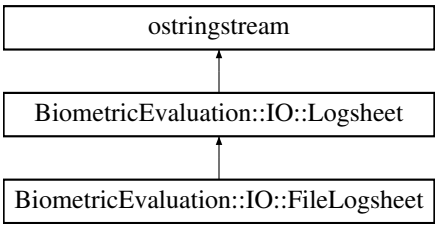
An object pointer to the new log sheet.

Exceptions

Error::ObjectExists	The sheet was previously created.
Error::StrategyError	An error occurred when using the underlying file system.

H.41 BiometricEvaluation::IO::FileLogsheet Class Reference

A class to represent a single logging mechanism with a file as the backing store.
`#include <be_io_filelogsheet.h>`
Inheritance diagram for BiometricEvaluation::IO::FileLogsheet:



Public Member Functions

- FileLogsheet (const std::string &url, const std::string &description)

  Create a new log sheet.
- FileLogsheet (const std::string &url)

  Open an existing log sheet for appending.
- ~FileLogsheet ()
- std::string sequence (bool allEntries=false, bool trim=true, int32_t cursor=BE_FILELOGSHEET_SEQ_NEXT)

  Sequence through a FileLogsheet, returning one entry per invocation.
- void write (const std::string &entry)

  Write a string as an entry to the backing store.
- void writeComment (const std::string &entry)

  Write a string as a comment to the backing store.
- void writeDebug (const std::string &entry)

  Write a string as a debug entry to the backing store.
- void sync ()

  Synchronize any buffered data to the underlying backing store.

Static Public Member Functions

- static void mergeLogsheets (std::vector< std::shared_ptr< FileLogsheet >> &logsheets)

  Merge multiple FileLogsheets into a single FileLogsheet.
- static std::string trim (const std::string &entry)

  Trim delimiters from FileLogsheet entries.

Static Public Attributes

- static const int32_t BE_FILELOGSHEET_SEQ_START = 1
- static const int32_t BE_FILELOGSHEET_SEQ_NEXT = 2

Protected Member Functions

- FileLogsheet (const FileLogsheet &)
- FileLogsheet & operator= (const FileLogsheet &)
- void updateCursor ()

  Update the cursor position of the sequence file.

Protected Attributes

- std::auto_ptr< std::fstream > _theLogFile
- std::shared_ptr< std::fstream > _sequenceFile
- streamoff _cursor

Additional Inherited Members

H.41.1 Detailed Description

A class to represent a single logging mechanism with a file as the backing store.

A FileLogsheet object can be constructed and passed back to the client by the LogCabinet object. All sheets created in this manner are placed in a common area maintained by the cabinet.

H.41.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::FileLogsheet::FileLogsheet (const std::string & *url*, const std::string & *description*)

Create a new log sheet.

the log sheet is named by the uniform resource locator, usually starting with 'file:///'. However, relative and absolute path names are also accepted for backward compatibility.

Parameters

in	*url*	The Uniform Resource Locator of the FileLogsheet to be created.
in	*description*	The text used to describe the sheet. This text is written into the log file prior to any entries.

Exceptions

Error::ParameterError	The URL is malformed.
Error::ObjectExists	The sheet was previously created.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

BiometricEvaluation::IO::FileLogsheet::FileLogsheet (const std::string & *url*)

Open an existing log sheet for appending.

On open, the current entry counter is set to the last entry number plus one.

Note

Opening a large FileLogsheet may be a costly operation.

Parameters

in	*url*	The Uniform Resource Locator of the FileLogsheet to be opened.

Exceptions

Error::ParameterError	The URL is malformed.
Error::ObjectDoesNotExist	The sheet does not exist.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

BiometricEvaluation::IO::FileLogsheet::~FileLogsheet ( )

Destructor

BiometricEvaluation::IO::FileLogsheet::FileLogsheet (const FileLogsheet &) [protected]

Prevent copying of FileLogsheet objects

H.41.3 Member Function Documentation

static void BiometricEvaluation::IO::FileLogsheet::mergeLogsheets (std::vector< std::shared_ptr< FileLogsheet >> & *logsheets*) [static]

Merge multiple FileLogsheets into a single FileLogsheet.

Logsheet 2 - n will be appended to Logsheet 1.

Parameters

logSheets	Logsheet to merge.

Exceptions

Error::FileError	Error during log sequence.
Error::StrategyError	Error during log sequence.

FileLogsheet& BiometricEvaluation::IO::FileLogsheet::operator= (const FileLogsheet &) [protected]

Prevent copying of FileLogsheet objects

std::string BiometricEvaluation::IO::FileLogsheet::sequence (bool *allEntries* = `false`, bool *trim* = `true`, int32_t *cursor* = BE_FILELOGSHEET_SEQ_NEXT)

Sequence through a FileLogsheet, returning one entry per invocation.

Parameters

allEntries	Include debgug and comment entries when sequencing
trim	Whether or not to include entry delimiters.
cursor	The location within the sequence to return.

Returns

The contents of the sequenced entry, as was originally given to write().

Exceptions

Error::FileError,Error	occured while performing file IO.
Error::ObjectDoesNot↩ Exist	The FileLogsheet cannot be found on disk.
Error::StrategyError	Invalid cursor position or the contents of the FileLogsheet is malformed.

void BiometricEvaluation::IO::FileLogsheet::sync ( ) [virtual]

Synchronize any buffered data to the underlying backing store.

This syncing is dependent on the behavior of the underlying storage mechanism.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

Reimplemented from BiometricEvaluation::IO::Logsheet.

static std::string BiometricEvaluation::IO::FileLogsheet::trim (const std::string & *entry*) [static]

Trim delimiters from FileLogsheet entries.

Works for comments and numbered entries.

Parameters

in	*entry*	The entry to trim.

Returns

Delimiter-less entry.

void BiometricEvaluation::IO::FileLogsheet::updateCursor ( ) [protected]

Update the cursor position of the sequence file.

Exceptions

Error::FileError	Error getting file position from sequence file.

void BiometricEvaluation::IO::FileLogsheet::write (const std::string & *entry*) [virtual]

Write a string as an entry to the backing store.

This does not affect the current log entry buffer, but does increment the entry number.

Parameters

in	*entry*	The text of the log entry.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

Reimplemented from BiometricEvaluation::IO::Logsheet.

void BiometricEvaluation::IO::FileLogsheet::writeComment (const std::string & *entry*) [virtual]

Write a string as a comment to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

in	*entry*	The text of the comment.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

Reimplemented from BiometricEvaluation::IO::Logsheet.

void BiometricEvaluation::IO::FileLogsheet::writeDebug (const std::string & *entry*) [virtual]

Write a string as a debug entry to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A debug line is prefixed with DebugDelimiter followed by a space.

Parameters

in	*entry*	The text of the debug message.

Exceptions

Error::StrategyError	An error occurred when logging.

Reimplemented from BiometricEvaluation::IO::Logsheet.

H.41.4 Member Data Documentation

streamoff BiometricEvaluation::IO::FileLogsheet::_cursor [protected]

Position of the sequencer, relative to SOF

std::shared_ptr<std::fstream> BiometricEvaluation::IO::FileLogsheet::_sequenceFile [protected]

Stream used for sequencing

std::auto_ptr<std::fstream> BiometricEvaluation::IO::FileLogsheet::_theLogFile [protected]

Stream used for writing the log file

const int32_t BiometricEvaluation::IO::FileLogsheet::BE_FILELOGSHEET_SEQ_NEXT = 2 [static]

Sequence from current position

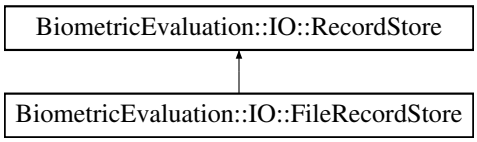
const int32_t BiometricEvaluation::IO::FileLogsheet::BE_FILELOGSHEET_SEQ_START = 1 [static]

Sequence from beginning

H.42 BiometricEvaluation::IO::FileRecordStore Class Reference

`#include <be_io_filerecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::FileRecordStore:



Public Member Functions

- FileRecordStore (const std::string &pathname, const std::string &description)
- FileRecordStore (const std::string &name, uint8_t mode=IO::READWRITE)
- uint64_t getSpaceUsed () const

  Obtain real storage utilization.
- void insert (const std::string &key, const void *const data, const uint64_t size)
- void remove (const std::string &key)

- uint64_t read (const std::string &key, void *const data) const
- virtual void replace (const std::string &key, const void *const data, const uint64_t size)
- virtual uint64_t length (const std::string &key) const
- void flush (const std::string &key) const
- uint64_t sequence (std::string &key, void *const data=nullptr, int cursor=BE_RECSTORE_SEQ_NEXT)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (const std::string &key)
- void move (const std::string &pathname)

  Move the RecordStore.
- **FileRecordStore** (const FileRecordStore &)=delete
- FileRecordStore & **operator=** (const FileRecordStore &)=delete

Protected Member Functions

- std::string **canonicalName** (const std::string &name) const

Additional Inherited Members

H.42.1 Detailed Description

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

Note

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

H.42.2 Constructor & Destructor Documentation

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

Create a new FileRecordStore, read/write mode.

Parameters

in	*pathname*	The directory where the store is to be created.
in	*description*	The store's description.

Exceptions

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

BiometricEvaluation::IO::FileRecordStore::FileRecordStore (const std::string & *name*, uint8_t *mode* = *IO::READWRITE*)

Open an existing FileRecordStore.

Parameters

in	*name*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

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

H.42.3 Member Function Documentation

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

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::FileRecordStore::getSpaceUsed ( ) const [virtual]

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::FileRecordStore::insert (const std::string & *key*, const void *const *data*, const uint64_t *size*) [virtual]

Insert a record into the store.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

virtual uint64_t BiometricEvaluation::IO::FileRecordStore::length (const std::string & *key*) const [virtual]

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Move the RecordStore.

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

Parameters

in	*pathname*	The new path of the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::FileRecordStore::read (const std::string & *key*, void *const *data*) const [virtual]

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

Parameters

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

Returns

The size of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Remove a record from the store.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Replace a complete record in a store.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::FileRecordStore::sequence (std::string & *key*, void *const *const *data* = *nullptr*, int *cursor* = BE_RECSTORE_SEQ_NEXT) [virtual]

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

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

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to nullptr to indicate only the key is wanted.

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

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::FileRecordStore::setCursorAtKey (const std::string & *key*) [virtual]

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

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

H.43 BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem Struct Reference

Representation of information about a fingerprint reader system.

`#include <be_feature_an2k7minutiae.h>`

Public Attributes

- std::string name
- EncodingMethod method
- std::string equipment

H.43.1 Detailed Description

Representation of information about a fingerprint reader system.

H.43.2 Member Data Documentation

std::string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::equipment

Optional ID for equipment used in system

EncodingMethod BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::method

Method used to encoded minutiae

std::string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::name

Name for system that encoded minutiae

H.44 BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition Struct Reference

Locations of an individual finger segment in a slap.

`#include <be_finger_an2kview_capture.h>`

Public Member Functions

- FingerSegmentPosition (const Finger::Position fingerPosition, const Image::CoordinateSet coordinates)

  Create an FingerSegmentPosition struct.

Public Attributes

- Finger::Position fingerPosition
- Image::CoordinateSet coordinates

H.44.1 Detailed Description

Locations of an individual finger segment in a slap.

H.44.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::FingerSegmentPosition (const Finger::Position *fingerPosition*, const Image::CoordinateSet *coordinates*)

Create an FingerSegmentPosition struct.

Parameters

fingerPosition	Finger depicted in this segment.
coordinates	Collection of coordinates that compose the segment bonding polygon.

H.44.3 Member Data Documentation

Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::coordinates

Points composing the segmented polygon

Finger::Position BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::fingerPosition

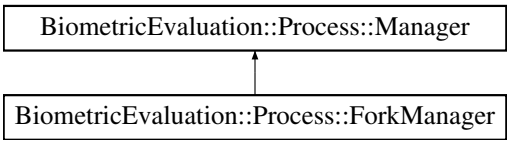
Finger depicted in this segment

H.45 BiometricEvaluation::Process::ForkManager Class Reference

Manager implementation that starts Workers by calling fork(2).

`#include <be_process_forkmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::ForkManager:



Public Member Functions

- ForkManager ()
- std::shared_ptr< WorkerController > addWorker (std::shared_ptr< Worker > worker)

  Adds a Worker to be managed by this Manager.
- void startWorkers (bool wait=true, bool communicate=false)

  Begin Worker's work.
- void startWorker (std::shared_ptr< WorkerController > worker, bool wait=true, bool communicate=false)

  Start a worker.
- int32_t stopWorker (std::shared_ptr< WorkerController > workerController)

  Ask Worker to exit.
- void broadcastSignal (int signo)

  Send a POSIX signal to all workers.
- bool responsibleFor (const pid_t pid) const

  Obtain whether or not this ForkManager is responsbile for a particular PID.
- void setNotWorking (const pid_t pid)

  Set Status.isWorking for PID to false.
- void markAllFinished ()

  Call setNotWorking() for all PIDs known to this ForkManager.
- bool getIsWorkingStatus (const pid_t pid) const

  Get Status.isWorking for PID.
- void waitForWorkerExit ()

  Block until all Workers have exited.
- ~ForkManager ()

  ForkManager destructor.
- void setExitCallback (void(*exitCallback)(std::shared_ptr< ForkWorkerController > worker, int stat_loc))

  Call a function in your program when a child exits.

Static Public Member Functions

- static void defaultExitCallback (std::shared_ptr< ForkWorkerController > worker, int status)

  A default exit callback function.

Static Public Attributes

- static std::list< ForkManager * > FORKMANAGERS

  List of all instantiated ForkManagers.

Additional Inherited Members

H.45.1 Detailed Description

Manager implementation that starts Workers by calling fork(2).

H.45.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::ForkManager::ForkManager ( )

ForkManager constructor.

H.45.3 Member Function Documentation

std::shared_ptr<WorkerController> BiometricEvaluation::Process::ForkManager::addWorker (std::shared_ptr< Worker > *worker*) `[virtual]`

Adds a Worker to be managed by this Manager.
Parameters

worker	A Worker instance to run.

Returns

shared_ptr to worker.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::ForkManager::broadcastSignal (int *signo*)

Send a POSIX signal to all workers.
Parameters

in	*signo*	The signal to send.

static void BiometricEvaluation::Process::ForkManager::defaultExitCallback (std::shared_ptr< ForkWorkerController > *worker*, int *status*) `[static]`

A default exit callback function.
Writes to stdout in the form: PID #: Exited .
Parameters

worker	The ForkWorkerController object that exited.
status	The status of the Worker that exited (from wait(2)).

bool BiometricEvaluation::Process::ForkManager::getIsWorkingStatus (const pid_t *pid*) const

Get Status.isWorking for PID.

Parameters

in	*pid*	PID whose inWorking flag should be queried

Exceptions

Error::ObjectDoesNotExist	PID not under this manager's control.

bool BiometricEvaluation::Process::ForkManager::responsibleFor (const pid_t *pid*) const

Obtain whether or not this ForkManager is responsbile for a particular PID.

Parameters

in	*pid*	PID in question

Returns

true if this ForkManager spawned pid, false otherwise.

void BiometricEvaluation::Process::ForkManager::setExitCallback (void(∗)(std::shared_ptr< ForkWorkerController > worker, int stat_loc) *exitCallback*)

Call a function in your program when a child exits.

Parameters

exitCallback	Function pointer to a method that takes a shared_ptr to a ForkWorkerController and the integer status information.

Note

The exit callback will not have any effect if the Manager is not set to wait for Workers.

void BiometricEvaluation::Process::ForkManager::setNotWorking (const pid_t *pid*)

Set Status.isWorking for PID to false.

Parameters

in	*pid*	PID whose inWorking flag should be set to false

Exceptions

Error::ObjectDoesNotExist	PID not under this manager's control.

void BiometricEvaluation::Process::ForkManager::startWorker (std::shared_ptr< WorkerController > *worker*, bool *wait* = `true`, bool *communicate* = `false`) [virtual]

Start a worker.

Parameters

	worker	Pointer to a WorkerController that is being managed by this Manager instance.
	wait	Whether or not to wait for this Worker to exit before returning control to the caller.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::ForkManager::startWorkers (bool *wait* = `true`, bool *communicate* = `false`) `[virtual]`

Begin Worker's work.

Parameters

in	*wait*	Whether or not to wait for all Workers to return before returning.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	At least one Worker is already working.
Error::StrategyError	Problem forking.

Implements BiometricEvaluation::Process::Manager.

int32_t BiometricEvaluation::Process::ForkManager::stopWorker (std::shared_ptr< WorkerController > *workerController*) `[virtual]`

Ask Worker to exit.

Sends SIGUSR1 to the Worker, which ForkManager will handle automatically.

Parameters

workerController	Pointer to the ForkWorkerController that should be stopped.

Returns

Exit status of worker.

Exceptions

Error::ObjectDoesNotExist	worker is not working.
Error::StrategyError	Problem sending the signal.

Attention

Do not call stopWorker() when communication is enabled unless you will be finished with communication for all Workers at that point. This creates a race condition for reads()/writes() when the Worker exits.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::ForkManager::waitForWorkerExit ( ) `[virtual]`

Block until all Workers have exited.

Use this method if wait=false was set during a call to startWorker(s) but now wait=true is desired.

Implements BiometricEvaluation::Process::Manager.

H.45.4 Member Data Documentation

std::list<ForkManager*> BiometricEvaluation::Process::ForkManager::FORKMANAGERS `[static]`

List of all instantiated ForkManagers.

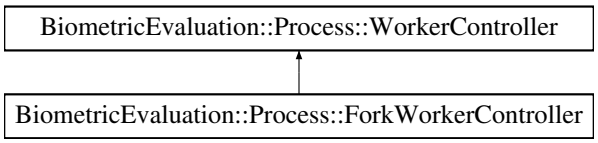
This is not a list of managed pointers to ForkManagers. If it was, the smart pointer's destructor would attempt to delete the object being pointed to at program termination, which is ultimately sometime after the destructor of the ForkManager itself was called.

H.46 BiometricEvaluation::Process::ForkWorkerController Class Reference

Wrapper of a Worker returned from a Process::ForkManager.

`#include <be_process_forkmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::ForkWorkerController:



Public Member Functions

- bool isWorking () const

  Obtain whether or not Worker is working.
- bool everWorked () const

  Obtain whether or not this Worker has ever worked.
- void reset ()

  Reuse the Worker.
- pid_t getPID () const

  Obtain the PID of this process this instance represents.
- ~ForkWorkerController ()

  ForkWorkerController destructor.

Static Public Member Functions

- static void _stop (int signal)

  Tell _staticWorker to stop.

Friends

- void ForkManager::startWorkers (bool wait, bool communicate)

  Begin Worker's work.
- void ForkManager::startWorker (std::shared_ptr< WorkerController > worker, bool wait, bool communicate)

  Restart a completed Worker.
- int32_t ForkManager::stopWorker (std::shared_ptr< WorkerController > workerController)

  Ask Worker to exit.
- std::shared_ptr< WorkerController > ForkManager::addWorker (std::shared_ptr< Worker > worker)

  Adds a Worker to be managed by this Manager.

Additional Inherited Members

H.46.1 Detailed Description

Wrapper of a Worker returned from a Process::ForkManager.

H.46.2 Member Function Documentation

static void BiometricEvaluation::Process::ForkWorkerController::_stop (int *signal*) `[static]`

Tell _staticWorker to stop.

Called by the child process instance when SIGUSR1 is received.

Parameters

signal	The signal caught that prompted this function to be called (SIGUSR1).

bool BiometricEvaluation::Process::ForkWorkerController::everWorked ( ) const `[virtual]`

Obtain whether or not this Worker has ever worked.

Returns

true the Worker has ever or is currently working, false otherwise.

Note

reset() will change the result of this method.

Implements BiometricEvaluation::Process::WorkerController.

pid_t BiometricEvaluation::Process::ForkWorkerController::getPID ( ) const

Obtain the PID of this process this instance represents.

Returns

pid of the process this instance represents.

Note

Call isRunning() before doing anything with the PID returned from this function.

bool BiometricEvaluation::Process::ForkWorkerController::isWorking ( ) const [virtual]

Obtain whether or not Worker is working.

Returns

Whether or not the Worker is working.

Implements BiometricEvaluation::Process::WorkerController.

void BiometricEvaluation::Process::ForkWorkerController::reset ( ) [virtual]

Reuse the Worker.
Exceptions

Error::ObjectExists	The previously started Worker is still running.

Reimplemented from BiometricEvaluation::Process::WorkerController.

H.46.3 Friends And Related Function Documentation

std::shared_ptr<WorkerController> ForkManager::addWorker (std::shared_ptr< Worker > *worker*) [friend]

Adds a Worker to be managed by this Manager.
Parameters

worker	A Worker instance to run.

Returns

shared_ptr to worker.

void ForkManager::startWorker (std::shared_ptr< WorkerController > *worker*, bool *wait*, bool *communicate*) [friend]

Restart a completed Worker.
Parameters

	worker	Pointer to a WorkerController that is being managed by this Manager instance.
	wait	Whether or not to wait for this Worker to exit before returning control to the caller.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

void ForkManager::startWorkers (bool *wait*, bool *communicate*) [friend]

Begin Worker's work.

Parameters

in	*wait*	Whether or not to wait for all Workers to return before returning.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	One or more of the Workers is already working.
Error::StrategyError	Problem forking.

int32_t ForkManager::stopWorker (std::shared_ptr< WorkerController > *workerController*) [friend]

Ask Worker to exit.

Sends SIGUSR1 to the Worker, which ForkManager will handle automatically.

Parameters

workerController	Pointer to the ForkWorkerController that should be stopped.

Returns

Exit status of worker.

Exceptions

Error::ObjectDoesNotExist	worker is not working.
Error::StrategyError	Problem sending the signal.

H.47 BiometricEvaluation::Video::Frame Struct Reference

Public Attributes

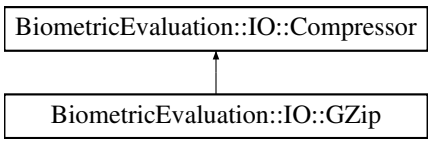
- Image::Size **size**
- int64_t **timestamp**
- Memory::uint8Array **data**

H.48 BiometricEvaluation::IO::GZip Class Reference

Compressor for gzip compression from zlib.

`#include <be_io_gzip.h>`

Inheritance diagram for BiometricEvaluation::IO::GZip:



Public Member Functions

- Memory::uint8Array compress (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize) const

  Compress a buffer.
- Memory::uint8Array compress (const Memory::uint8Array &uncompressedData) const

  Compress a buffer.
- void compress (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize, const std::string &outputFile) const

  Compress a buffer.
- void compress (const Memory::uint8Array &uncompressedData, const std::string &outputFile) const

  Compress a buffer.
- Memory::uint8Array compress (const std::string &inputFile) const

  Compress a file.
- void compress (const std::string &inputFile, const std::string &outputFile) const

  Compress a file.
- Memory::uint8Array decompress (const uint8_t *const compressedData, uint64_t compressedDataSize) const

  Decompress a compressed buffer.
- Memory::uint8Array decompress (const Memory::uint8Array &compressedData) const

  Decompress a compressed buffer.
- Memory::uint8Array decompress (const std::string &input) const

  Decompress a compressed buffer into a file.
- void decompress (const std::string &inputFile, const std::string &outputFile) const

  Decompress a file.
- void decompress (const uint8_t *const compressedData, const uint64_t compressedDataSize, const std::string &outputFile) const

  Decompress a file.
- void decompress (const Memory::uint8Array &compressedData, const std::string &outputFile) const

  Decompress a file.
- GZip (const GZip &other)=delete

  Copy constructor (disabled).
- GZip & operator= (const GZip &other)=delete

  Assignment overload (disabled).

Static Public Attributes

- static const std::string COMPRESSION_LEVEL
- static const std::string COMPRESSION_STRATEGY
- static const std::string COMPRESSION_METHOD
- static const std::string INPUT_DATA_TYPE
- static const std::string WINDOW_BITS
- static const std::string MEMORY_LEVEL
- static const std::string CHUNK_SIZE

Additional Inherited Members

H.48.1 Detailed Description

Compressor for gzip compression from zlib.

H.48.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::GZip::GZip (const GZip & *other*) [delete]

Copy constructor (disabled).

Disabled because Properties member of parent cannot be copied.

Parameters

other	GZip to copy.

H.48.3 Member Function Documentation

Memory::uint8Array BiometricEvaluation::IO::GZip::compress (const uint8_t *const *uncompressedData,* uint64_t *uncompressedDataSize*) const [virtual]

Compress a buffer.

Parameters

uncompressed↩ Data	Uncompressed data buffer to compress.
uncompressed↩ DataSize	Size of uncompressedData.

Returns

Compressed buffer.

Exceptions

Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::compress (const Memory::uint8Array & *uncompressedData*) const [virtual]

Compress a buffer.

Parameters

uncompressed↩ Data	Uncompressed data buffer to compress.

Returns

Compressed buffer.

Exceptions

Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::compress (const uint8_t *const *uncompressedData,* uint64_t *uncompressedDataSize,* const std::string & *outputFile*) const [virtual]

Compress a buffer.
Parameters

uncompressed↩ Data	Uncompressed data buffer to compress.
uncompressed↩ DataSize	Size of uncompressedData.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::compress (const Memory::uint8Array & *uncompressedData,* const std::string & *outputFile*) const [virtual]

Compress a buffer.
Parameters

uncompressed↩ Data	Uncompressed data buffer to compress.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::compress (const std::string & *inputFile*) const [virtual]

Compress a file.
Parameters

inputFile	Path to file to compress.

Returns

Compressed buffer.

Exceptions

Error::ObjectDoesNot↩ Exist	Input file does not exist.
Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::compress (const std::string & *inputFile*, const std::string & *outputFile*) const [virtual]

Compress a file.
Parameters

inputFile	Path to file to compress.
outputFile	Path to location where compressed version will be saved.

Exceptions

Error::ObjectDoesNot↩ Exist	Input file does not exist.
Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (const uint8_t *const *compressedData*, uint64_t *compressedDataSize*) const [virtual]

Decompress a compressed buffer.
Parameters

compressed↩ Data	Compressed data buffer to decompress.
compressed↩ DataSize	Size of compressedData.

Returns

Decompressed data.

Exceptions

Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (const Memory::uint8Array & *compressedData*) const [virtual]

Decompress a compressed buffer.
Parameters

compressed↩ Data	Compressed data buffer to decompress.

Returns

Decompressed data.

Exceptions

Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (const std::string & *inputFile*) const `[virtual]`

Decompress a compressed buffer into a file.

Parameters

inputFile	Location to save compressed file.

Returns

Decompressed data.

Exceptions

Error::StrategyError	Error in decompression unit.
Error::ObjectDoesNot↩ Exists	Output file already exists.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::decompress (const std::string & *inputFile*, const std::string & *outputFile*) const `[virtual]`

Decompress a file.

Parameters

inputFile	Path to file to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectDoesNot↩ Exist	Input file does not exist.
Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::decompress (const uint8_t *const *compressedData*, const uint64_t *compressedDataSize*, const std::string & *outputFile*) const `[virtual]`

Decompress a file.

Parameters

compressed↩ Data	Compressed data buffer to decompress.
compressed↩ DataSize	Size of compressedData.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::decompress (const Memory::uint8Array & *compressedData*, const std::string & *outputFile*) const [virtual]

Decompress a file.

Parameters

compressed↩ Data	Compressed data buffer to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

GZip& BiometricEvaluation::IO::GZip::operator= (const GZip & *other*) [delete]

Assignment overload (disabled).

Disabled because Properties member of parent cannot be assigned.

Parameters

other	GZip to assign.

Returns

lhs GZip.

H.48.4 Member Data Documentation

const std::string BiometricEvaluation::IO::GZip::CHUNK_SIZE [static]

How many bytes to work at a time

const std::string BiometricEvaluation::IO::GZip::COMPRESSION_LEVEL [static]

How thorough the compression should be

const std::string BiometricEvaluation::IO::GZip::COMPRESSION_METHOD [static]

Which underlying method in the compressor

const std::string BiometricEvaluation::IO::GZip::COMPRESSION_STRATEGY [static]

Which underlying algorithm to use

const std::string BiometricEvaluation::IO::GZip::INPUT_DATA_TYPE [static]

The type of data being compressed

const std::string BiometricEvaluation::IO::GZip::MEMORY_LEVEL [static]

How much memory for internal compression state

const std::string BiometricEvaluation::IO::GZip::WINDOW_BITS [static]

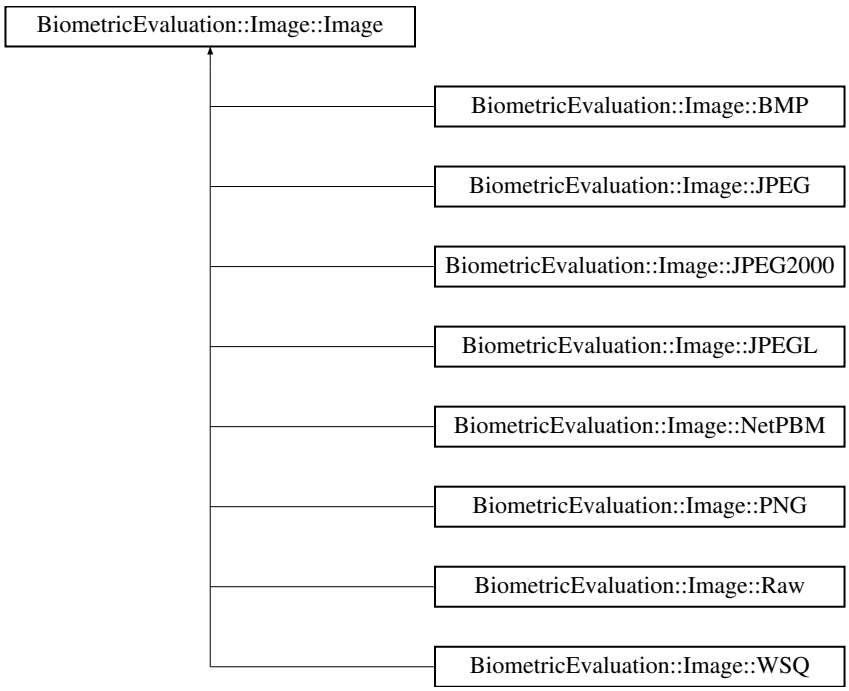
Window size

H.49 BiometricEvaluation::Image::Image Class Reference

Represent attributes common to all images.

`#include <be_image_image.h>`

Inheritance diagram for BiometricEvaluation::Image::Image:



Public Member Functions

- Image (const uint8_t *data, const uint64_t size, const Size dimensions, const uint32_t depth, const Resolution resolution, const CompressionAlgorithm compression)

  Parent constructor for all Image classes.
- Image (const uint8_t *data, const uint64_t size, const CompressionAlgorithm compression)

Parent constructor for all Image classes.

- CompressionAlgorithm getCompressionAlgorithm () const

  Accessor for the CompressionAlgorithm of the image.
- Resolution getResolution () const

  Accessor for the resolution of the image.
- Memory::uint8Array getData () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual Memory::uint8Array getRawData () const =0

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- virtual Memory::uint8Array getRawGrayscaleData (uint8_t depth=8) const =0

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

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

  Accessor for the color depth of the image in bits.

Static Public Member Functions

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

  Calculate an equivalent color value for a color in an alternate colorspace.
- static std::shared_ptr< Image > openImage (const uint8_t *data, const uint64_t size)

  Determine the image type of a buffer of image data and create an Image object.
- static std::shared_ptr< Image > openImage (const Memory::uint8Array &data)

  Determine the image type of a buffer of image data and create an Image object.
- static std::shared_ptr< Image > openImage (const std::string &path)

  Determine the image type of an image file and create an Image object.
- static CompressionAlgorithm getCompressionAlgorithm (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.
- static CompressionAlgorithm getCompressionAlgorithm (const Memory::uint8Array &data)

  Determine the compression algorithm of a buffer of image data.
- static CompressionAlgorithm getCompressionAlgorithm (const std::string &path)

  Determine the compression algorithm of a file.

Static Public Attributes

- static const uint32_t bitsPerComponent = 8

Protected Member Functions

- void setResolution (const Resolution resolution)

  Mutator for the resolution of the image .
- void setDimensions (const Size dimensions)

  Mutator for the dimensions of the image in pixels.
- void setDepth (const uint32_t depth)

  Mutator for the color depth of the image in bits.
- const uint8_t * getDataPointer () const
- uint64_t getDataSize () const

H.49.1 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.

H.49.2 Constructor & Destructor Documentation

BiometricEvaluation::Image::Image::Image (const uint8_t * *data*, const uint64_t *size*, const Size *dimensions*, const uint32_t *depth*, const Resolution *resolution*, const CompressionAlgorithm *compression*)

Parent constructor for all Image classes.

Parameters

in	*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	*depth*	The image depth, in bits-per-pixel.
in	*resolution*	The resolution of the image
in	*compression*	The CompressionAlgorithm of data.

Exceptions

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

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.

H.49.3 Member Function Documentation

CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm ( ) const

Accessor for the CompressionAlgorithm of the image.

Returns

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

static CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm (const uint8_t * *data*, const uint64_t *size*) [static]

Determine the compression algorithm of a buffer of image data.

Parameters

in	*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.

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.

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.

Memory::uint8Array BiometricEvaluation::Image::Image::getData ( ) const

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

Returns

AutoArray holding image data.

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

Returns

Const pointer to buffer underlying _data.

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

Returns

Size of _data.

uint32_t BiometricEvaluation::Image::Image::getDepth ( ) const

Accessor for the color depth of the image in bits.

Returns

The color depth of the image (bit).

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

Accessor for the dimensions of the image in pixels.

Returns

Coordinate object containing dimensions in pixels.

virtual Memory::uint8Array BiometricEvaluation::Image::Image::getRawData ( ) const [pure virtual]

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

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError	Error decompressing image data.

Implemented in BiometricEvaluation::Image::NetPBM, BiometricEvaluation::Image::JPEG2000, BiometricEvaluation::Image::JPEG, BiometricEvaluation::Image::PNG, BiometricEvaluation::Image::Raw, BiometricEvaluation::Image::BMP, BiometricEvaluation::Image::JPEGL, and BiometricEvaluation::Image::WSQ.

virtual Memory::uint8Array BiometricEvaluation::Image::Image::getRawGrayscaleData (uint8_t *depth* = *8*) const [pure virtual]

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

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

Note

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

Implemented in BiometricEvaluation::Image::NetPBM, BiometricEvaluation::Image::JPEG2000, BiometricEvaluation::Image::PNG, BiometricEvaluation::Image::JPEG, BiometricEvaluation::Image::Raw, BiometricEvaluation::Image::BMP, BiometricEvaluation::Image::WSQ, and BiometricEvaluation::Image::JPEGL.

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

Accessor for the resolution of the image.

Returns

Resolution struct

static std::shared_ptr<Image> BiometricEvaluation::Image::Image::openImage (const uint8_t * *data*, const uint64_t *size*) [static]

Determine the image type of a buffer of image data and create an Image object.

Parameters

in	*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.

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.

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.

void BiometricEvaluation::Image::Image::setDepth (const uint32_t *depth*) [protected]

Mutator for the color depth of the image in bits.
Parameters

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

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).

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

Mutator for the resolution of the image .

Parameters

in	*resolution*	Resolution struct.

static uint64_t BiometricEvaluation::Image::Image::valueInColorspace (uint64_t *color*, uint64_t *maxColorValue*, uint8_t *depth*) [static]

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

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.

H.49.4 Member Data Documentation

const uint32_t BiometricEvaluation::Image::Image::bitsPerComponent = 8 [static]

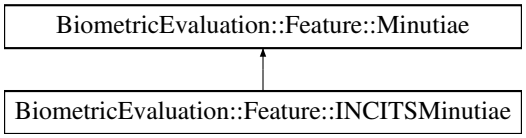
Number of bits per color component

H.50 BiometricEvaluation::Feature::INCITSMinutiae Class Reference

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

`#include <be_feature_incitsminutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::INCITSMinutiae:



Public Member Functions

- MinutiaeFormat getFormat () const

  Obtain the minutiae format kind.
- MinutiaPointSet getMinutiaPoints () const

  Obtain the set of finger minutiae data points. The set may be empty.
- RidgeCountItemSet getRidgeCountItems () const

  Obtain the set of ridge count data items. The set may be empty.
- CorePointSet getCores () const

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

  Obtains the set of delta positions. The set may be empty.
- INCITSMinutiae (const MinutiaPointSet &mps, const RidgeCountItemSet &rcis, const CorePointSet &cps, const DeltaPointSet &dps)

Construct an INCITS Minutiae object from its components.

- INCITSMinutiae ()

  Default constructor for an INCITS Minutiae object.
- void setMinutiaPoints (const MinutiaPointSet &mps)

  Mutator for the minutiae point set.
- void setRidgeCountItems (const RidgeCountItemSet &rcis)

  Mutator for the ridge count items.
- void setCorePointSet (const CorePointSet &cps)

  Mutator for the set of core points.
- void setDeltaPointSet (const DeltaPointSet &dps)

  Mutator for the set of delta points.

Static Public Attributes

- static const std::string **FMR_ANSI_SPEC_VERSION**
- static const std::string **FMR_ISO_SPEC_VERSION**
- static const std::string **FMR_ANSI07_SPEC_VERSION**
- static const uint8_t **FMR_SPEC_VERSION_LEN** = 4
- static const uint32_t **FED_HEADER_LENGTH** = 4
- static const uint32_t **FED_RCD_ITEM_LENGTH** = 3
- static const uint16_t **FMD_MINUTIA_TYPE_MASK** = 0xC000
- static const uint16_t **FMD_RESERVED_MASK** = 0xC000
- static const uint16_t **FMD_MINUTIA_TYPE_SHIFT** = 14
- static const uint16_t **FMD_RESERVED_SHIFT** = 14
- static const uint16_t **FMD_X_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_Y_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_MASK** = 0xC0
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_SHIFT** = 6
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_ANGLE_MASK** = 0x3F
- static const uint16_t **FMD_MIN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MAX_MINUTIA_QUALITY** = 100
- static const uint16_t **FMD_UNKNOWN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MIN_MINUTIA_ANGLE** = 0
- static const uint16_t **FMD_MAX_MINUTIA_ANGLE** = 179
- static const uint16_t **FMD_MAX_MINUTIA_ISONC_ANGLE** = 255
- static const uint16_t **FMD_MAX_MINUTIA_ISOCC_ANGLE** = 63
- static const uint16_t **FMD_ANSI_ANGLE_UNIT** = 2
- static const uint16_t **FMD_ISO_ANGLE_UNIT**
- static const uint16_t **FMD_ISOCC_ANGLE_UNIT**
- static const uint16_t **FMD_MINUTIA_TYPE_OTHER** = 0
- static const uint16_t **FMD_MINUTIA_TYPE_RIDGE_ENDING** = 1
- static const uint16_t **FMD_MINUTIA_TYPE_BIFURCATION** = 2
- static const uint16_t **FMR_MIN_FINGER_QUALITY** = 0
- static const uint16_t **FMR_MAX_FINGER_QUALITY** = 100
- static const uint16_t **ISO_UNKNOWN_FINGER_QUALITY** = 0
- static const uint16_t **FED_RESERVED** = 0x0000
- static const uint16_t **FED_RIDGE_COUNT** = 0x0001
- static const uint16_t **FED_CORE_AND_DELTA** = 0x0002

- static const uint16_t **RCE_NONSPECIFIC** = 0x00
- static const uint16_t **RCE_FOUR_NEIGHBOR** = 0x01
- static const uint16_t **RCE_EIGHT_NEIGHBOR** = 0x02
- static const uint16_t **CORE_TYPE_NONANGULAR** = 0x00
- static const uint16_t **CORE_TYPE_ANGULAR** = 0x01
- static const uint16_t **DELTA_TYPE_NONANGULAR** = 0x00
- static const uint16_t **DELTA_TYPE_ANGULAR** = 0x01

H.50.1 Detailed Description

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

The base INCTISMinutiae class is responsible for reading minutiae data points and extended data. Each minutiae point, ridge count item, core, and delta is represented in the native ANSI/INCITS format. Objects of this base class cannot be instantiated, but rather derived classes are used to represent minutiae data taken from the INCITS-derived record formats.

H.50.2 Constructor & Destructor Documentation

BiometricEvaluation::Feature::INCITSMinutiae::INCITSMinutiae (const MinutiaPointSet & *mps,* const RidgeCountItemSet & *rcis,* const CorePointSet & *cps,* const DeltaPointSet & *dps*)

Construct an INCITS Minutiae object from its components.

The buffer index must be set to the location in the buffer to start reading minutiae data points and extended data.

Parameters

in	*mps*	The set of minutiae points.
in	*rcis*	The set of ridge count items.
in	*cps*	The set of core points.
in	*dps*	The set of delta points.

H.50.3 Member Function Documentation

void BiometricEvaluation::Feature::INCITSMinutiae::setCorePointSet (const CorePointSet & *cps*)

Mutator for the set of core points.

Parameters

in	*cps*	The set of core points.

void BiometricEvaluation::Feature::INCITSMinutiae::setDeltaPointSet (const DeltaPointSet & *dps*)

Mutator for the set of delta points.

Parameters

in	*dps*	The set of delta point items.

void BiometricEvaluation::Feature::INCITSMinutiae::setMinutiaPoints (const MinutiaPointSet & *mps*)

Mutator for the minutiae point set.

Parameters

in	*mps*	The minutiae points.

void BiometricEvaluation::Feature::INCITSMinutiae::setRidgeCountItems (const RidgeCountItemSet & *rcis*)

Mutator for the ridge count items.
Parameters

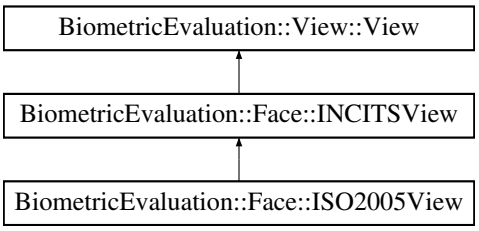
in	*rcis*	The set of ridge count items.

H.51 BiometricEvaluation::Face::INCITSView Class Reference

A class to represent single facial image view and derived information.

`#include <be_face_incitsview.h>`

Inheritance diagram for BiometricEvaluation::Face::INCITSView:



Public Member Functions

- Face::Gender getGender () const

  Obtain the gender.
- Face::EyeColor getEyeColor () const

  Obtain the eye color.
- Face::HairColor getHairColor () const

  Obtain the hair color.
- bool propertiesConsidered () const

  Indicate whether properties are specified.
- void getPropertySet (Face::PropertySet &propertySet) const

  Get the set of properties.
- BiometricEvaluation::Face::Expression **getExpression** () const
- void getFeaturePointSet (BiometricEvaluation::Feature::MPEGFacePointSet &featurePointSet) const

  Obtain the set of.
- Face::ImageType getImageType () const

  Obtain the face image type.
- Face::ImageDataType getImageDataType () const

  Obtain the face image data type.
- Face::PoseAngle getPoseAngle () const

  Obtain the face pose angle.

- Face::ColorSpace getColorSpace () const

  Obtain the color space.
- Face::SourceType getSourceType () const

  Obtain the source type.
- uint16_t getDeviceType () const

  Obtain the device type.

Protected Member Functions

- INCITSView (const std::string &filename, const uint32_t viewNumber)

  Construct the common components of an INCITS face view from records contained in files.
- INCITSView (const Memory::uint8Array &buffer, const uint32_t viewNumber)

  Construct an INCITS face view from a record contained in a buffer.
- Memory::uint8Array const & getFIDData () const

  Obtain a reference to the face image record data buffer.
- virtual void readHeader (BiometricEvaluation::Memory::IndexedBuffer &buf, const uint32_t format↩ Standard)

  Read the common face image data record header from an INCITS record, excepting the format identifier and version number data items.
- virtual void readFaceView (Memory::IndexedBuffer &buf)

  Read the common face representation information from an INCITS record.

Static Protected Attributes

- static const uint32_t **ISO2005_STANDARD** = 1
- static const uint32_t **BASE_FORMAT_ID** = 0x46414300

H.51.1 Detailed Description

A class to represent single facial image view and derived information.

A base Face::INCITSView class represents an INCITS/ANSI or ISO face view. This class defines the common interface for all ANSI/ISO views as well as common implementations. Subclasses specialize this class in order to represent other versions of the ANSI/ISO specs. Objects of this class cannot be created.

H.51.2 Constructor & Destructor Documentation

BiometricEvaluation::Face::INCITSView::INCITSView (const std::string & *filename*, const uint32_t *viewNumber*) [protected]

Construct the common components of an INCITS face view from records contained in files.

See documentation in child classes of INCITS for information on constructing INCITS-derived face views.

Parameters

in	*filename*	The name of the file containing the complete face image data record.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError	Invalid record format.
Error::FileError	Could not open or read from file.

BiometricEvaluation::Face::INCITSView::INCITSView (const Memory::uint8Array & *buffer*, const uint32_t *viewNumber*) [protected]

Construct an INCITS face view from a record contained in a buffer.

See documentation in child classes of INCITS for information on constructing INCITS-derived face views.

Parameters

in	*buffer*	The buffer containing the complete face image data record.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError	Invalid record format.

H.51.3 Member Function Documentation

Face::ColorSpace BiometricEvaluation::Face::INCITSView::getColorSpace ( ) const

Obtain the color space.

Returns

The color space code.

uint16_t BiometricEvaluation::Face::INCITSView::getDeviceType ( ) const

Obtain the device type.

Returns

The device type vendor code.

Face::EyeColor BiometricEvaluation::Face::INCITSView::getEyeColor ( ) const

Obtain the eye color.

Returns

The eye color code.

void BiometricEvaluation::Face::INCITSView::getFeaturePointSet (BiometricEvaluation::Feature::MPEGFacePointSet & *featurePointSet*) const

Obtain the set of.

Parameters

out	*featurePointSet*	The set of feature points.

Memory::uint8Array const& BiometricEvaluation::Face::INCITSView::getFIDData ( ) const [protected]

Obtain a reference to the face image record data buffer.

Returns

The entire face image record data.

Face::Gender BiometricEvaluation::Face::INCITSView::getGender ( ) const

Obtain the gender.

Returns

The gender code.

Face::HairColor BiometricEvaluation::Face::INCITSView::getHairColor ( ) const

Obtain the hair color.

Returns

The hair color code.

Face::ImageDataType BiometricEvaluation::Face::INCITSView::getImageDataType ( ) const

Obtain the face image data type.

Returns

The image data type.

Face::ImageType BiometricEvaluation::Face::INCITSView::getImageType ( ) const

Obtain the face image type.

Returns

The image type.

Face::PoseAngle BiometricEvaluation::Face::INCITSView::getPoseAngle ( ) const

Obtain the face pose angle.

Returns

The pose angle.

void BiometricEvaluation::Face::INCITSView::getPropertySet (Face::PropertySet & *propertySet*) const

Get the set of properties.

Returns

The set of properties.

Face::SourceType BiometricEvaluation::Face::INCITSView::getSourceType ( ) const

Obtain the source type.

Returns

The source type code.

bool BiometricEvaluation::Face::INCITSView::propertiesConsidered ( ) const

Indicate whether properties are specified.

Returns

true if properties are specified, false otherwise.

virtual void BiometricEvaluation::Face::INCITSView::readFaceView (Memory::IndexedBuffer & *buf*) [protected], [virtual]

Read the common face representation information from an INCITS record.

An Face representation from an INCITS record includes image information, gender, pose angle, etc.

Parameters

in,out	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the Facial information record.

Exceptions

DataError	The INCITS record has invalid or missing data.

virtual void BiometricEvaluation::Face::INCITSView::readHeader (BiometricEvaluation↩ ::Memory::IndexedBuffer & *buf,* const uint32_t *formatStandard*) [protected], [virtual]

Read the common face image data record header from an INCITS record, excepting the format identifier and version number data items.

Parameters

in	*buf*	The indexed buffer containing the record data, with the index starting at the first octet after the format identifier and version number data items. The index of the buffer will be changed to the location after the header.

in	*formatStandard*	Value indicating which header version to read; must be ISO2005_STAN↩ DARD

Exceptions

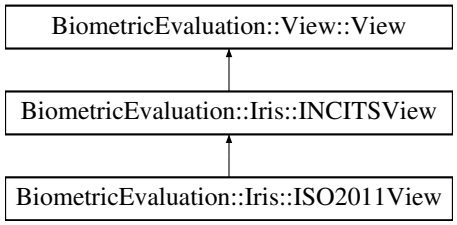
ParameterError	The formatStandard parameter is incorrect.
DataError	The INCITS record has invalid or missing data.

H.52 BiometricEvaluation::Iris::INCITSView Class Reference

A class to represent single iris view and derived information.

`#include <be_iris_incitsview.h>`

Inheritance diagram for BiometricEvaluation::Iris::INCITSView:



Classes

- struct QualitySubBlock

  Representation of an iris quality block.

Public Types

- typedef std::vector < QualitySubBlock > **QualitySet**

Public Member Functions

- uint8_t getCertificationFlag () const

  Obtain the certification flag.
- std::string getCaptureDateString () const

  Obtain the capture date as a string.
- Iris::CaptureDeviceTechnology getCaptureDeviceTechnology () const

  Obtain the capture device technology.
- uint16_t getCaptureDeviceVendor () const

  Obtain the capture device vendor.
- uint16_t getCaptureDeviceType () const

  Obtain the capture device type.
- void getQualitySet (Iris::INCITSView::QualitySet &qualitySet) const

  Obtain the set of quality sub-blocks.
- Iris::EyeLabel getEyeLabel () const

  Obtain the eye label type.

- Iris::ImageType getImageType () const

  Obtain the iris image type.
- void getImageProperties (BiometricEvaluation::Iris::Orientation &horizontalOrientation, BiometricEvaluation::Iris::Orientation &verticalOrientation, BiometricEvaluation::Iris::ImageCompression &compressionHistory) const

  Obtain the iris image properties.
- uint16_t getCameraRange ()

  Obtain the camera range.
- void getRollAngleInfo (uint16_t &rollAngle, uint16_t &rollAngleUncertainty)

  Obtain the roll angle information.
- void getIrisCenterInfo (uint16_t &irisCenterSmallestX, uint16_t &irisCenterSmallestY, uint16_t &irisCenterLargestX, uint16_t &irisCenterLargestY, uint16_t &irisDiameterSmallest, uint16_t &irisDiameterLargest)

  Obtain the iris center information. COORDINATE_UNDEF may be returned for any of the out parameters.

Static Public Attributes

- static const uint16_t **RANGE_UNASSIGNED** = 0
- static const uint16_t **RANGE_FAILED** = 1
- static const uint16_t **RANGE_OVERFLOW** = 65535
- static const uint16_t **ROLL_ANGLE_UNDEF** = 65535
- static const uint16_t **ROLL_UNCERTAIN_UNDEF** = 65535
- static const uint16_t **COORDINATE_UNDEF** = 0

Protected Member Functions

- INCITSView (const std::string &filename, const uint32_t viewNumber)

  Construct the common components of an INCITS iris view from records contained in files.
- INCITSView (const Memory::uint8Array &buffer, const uint32_t viewNumber)

  Construct an INCITS iris view from a record contained in a buffer.
- Memory::uint8Array const & getIIRData () const

  Obtain a reference to the iris image record data buffer.
- virtual void readHeader (BiometricEvaluation::Memory::IndexedBuffer &buf, const uint32_t formatStandard)

  Read the common iris image record header from an INCITS record, excepting the format identifier and version number data items.
- virtual void readIrisView (Memory::IndexedBuffer &buf)

  Read the common iris representation information from an INCITS record.

Static Protected Attributes

- static const uint32_t **ISO2011_STANDARD** = 1
- static const uint32_t **BASE_FORMAT_ID** = 0x49495200
- static const uint8_t **CAPTURE_DATE_LENGTH** = 9

H.52.1 Detailed Description

A class to represent single iris view and derived information.

A base Iris::INCITSView class represents an INCITS/ANSI or ISO iris view. This class defines the common interface for all ANSI/ISO views as well as common implementations. Subclasses specialize this class in order to represent other versions of the ANSI/ISO specs. Objects of this class cannot be created.

H.52.2 Constructor & Destructor Documentation

BiometricEvaluation::Iris::INCITSView::INCITSView (const std::string & *filename*, const uint32_t *viewNumber*) [protected]

Construct the common components of an INCITS iris view from records contained in files.

See documentation in child classes of INCITS for information on constructing INCITS-derived iris views.

Parameters

in	*filename*	The name of the file containing the complete iris image record.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError	Invalid record format.
Error::FileError	Could not open or read from file.

BiometricEvaluation::Iris::INCITSView::INCITSView (const Memory::uint8Array & *buffer*, const uint32_t *viewNumber*) [protected]

Construct an INCITS iris view from a record contained in a buffer.

See documentation in child classes of INCITS for information on constructing INCITS-derived iris views.

Parameters

in	*buffer*	The buffer containing the complete iris image record.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError	Invalid record format.

H.52.3 Member Function Documentation

uint16_t BiometricEvaluation::Iris::INCITSView::getCameraRange ( )

Obtain the camera range.

RANGE_UNASSIGNED, RANGE_FAILED, or RANGE_OVERFLOW may be returned.

Returns

The camera range.

std::string BiometricEvaluation::Iris::INCITSView::getCaptureDateString ( ) const

Obtain the capture date as a string.

Returns

The capture data and time.

Iris::CaptureDeviceTechnology BiometricEvaluation::Iris::INCITSView::getCaptureDeviceTechnology ( ) const

Obtain the capture device technology.

Returns

The capture device technology identifer.

uint16_t BiometricEvaluation::Iris::INCITSView::getCaptureDeviceType ( ) const

Obtain the capture device type.

Returns

The capture device type ID.

uint16_t BiometricEvaluation::Iris::INCITSView::getCaptureDeviceVendor ( ) const

Obtain the capture device vendor.

Returns

The capture device vendor ID.

uint8_t BiometricEvaluation::Iris::INCITSView::getCertificationFlag ( ) const

Obtain the certification flag.

Returns

The certification flag.

Iris::EyeLabel BiometricEvaluation::Iris::INCITSView::getEyeLabel ( ) const

Obtain the eye label type.

Returns

The eye label.

Memory::uint8Array const& BiometricEvaluation::Iris::INCITSView::getIIRData ( ) const [protected]

Obtain a reference to the iris image record data buffer.

Returns

The entire iris image record data.

void BiometricEvaluation::Iris::INCITSView::getImageProperties (BiometricEvaluation::Iris::Orientation & *horizontalOrientation*, BiometricEvaluation::Iris::Orientation & *verticalOrientation*, BiometricEvaluation::Iris::ImageCompression & *compressionHistory*) const

Obtain the iris image properties.

Parameters

out	*horizontal↩ Orientation*	The horizontal orientation.
out	*vertical↩ Orientation*	The vertical orientation.
out	*compression↩ History*	The image compression history.

Iris::ImageType BiometricEvaluation::Iris::INCITSView::getImageType ( ) const

Obtain the iris image type.

Returns

The image type.

void BiometricEvaluation::Iris::INCITSView::getIrisCenterInfo (uint16_t & *irisCenterSmallestX*, uint16_t & *irisCenterSmallestY*, uint16_t & *irisCenterLargestX*, uint16_t & *irisCenterLargestY*, uint16_t & *irisDiameterSmallest*, uint16_t & *irisDiameterLargest*)

Obtain the iris center information. COORDINATE_UNDEF may be returned for any of the out parameters.
Parameters

out	*irisCenter↩ SmallestX*	Smallest expected iris center X coordinate in pixels.
out	*irisCenter↩ SmallestY*	Smallest expected iris center Y coordinate in pixels.
out	*irisCenter↩ LargestX*	Largest expected iris center X coordinate in pixels.
out	*irisCenter↩ LargestY*	Largest expected iris center Y coordinate in pixels.
out	*irisDiameter↩ Smallest*	Smallest expected iris diameter in pixels.
out	*irisDiameter↩ Largest*	Largest expected iris diameter in pixels.

void BiometricEvaluation::Iris::INCITSView::getQualitySet (Iris::INCITSView::QualitySet & *qualitySet*) const

Obtain the set of quality sub-blocks.
Parameters

out	*qualitySet*	The set of quality sub-blocks.

void BiometricEvaluation::Iris::INCITSView::getRollAngleInfo (uint16_t & *rollAngle*, uint16_t & *rollAngleUncertainty*)

Obtain the roll angle information.

Parameters

out	*rollAngle*	The roll angle.
out	*rollAngle*↩ *Uncertainty*	The roll angle uncertainty.

virtual void BiometricEvaluation::Iris::INCITSView::readHeader (BiometricEvaluation↩ ::Memory::IndexedBuffer & *buf*, const uint32_t *formatStandard*) [protected], [virtual]

Read the common iris image record header from an INCITS record, excepting the format identifier and version number data items.

Parameters

in	*buf*	The indexed buffer containing the record data, with the index starting at the first octet after the format identifier and version number data items. The index of the buffer will be changed to the location after the header.
in	*formatStandard*	Value indicating which header version to read; must be ISO2011_STAN↩ DARD

Exceptions

ParameterError	The specVersion parameter is incorrect.
DataError	The INCITS record has invalid or missing data.

virtual void BiometricEvaluation::Iris::INCITSView::readIrisView (Memory::IndexedBuffer & *buf*) [protected], [virtual]

Read the common iris representation information from an INCITS record.

An Iris Representation from an INCITS record includes image information, cropping information, etc.

Parameters

in,out	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the Iris Representation.

Exceptions

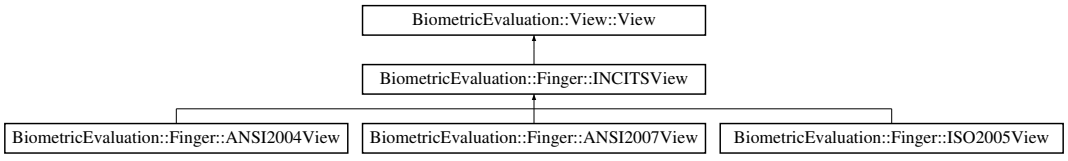
DataError	The INCITS record has invalid or missing data.

H.53 BiometricEvaluation::Finger::INCITSView Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_incitsview.h>`

Inheritance diagram for BiometricEvaluation::Finger::INCITSView:



Public Member Functions

- Feature::INCITSMinutiae getMinutiaeData () const

  Obtain the set of minutiae records.
- Finger::Position getPosition () const

  Obtain the finger position.
- Finger::Impression getImpressionType () const

  Obtain the finger impression code.
- uint32_t getQuality () const

  Obtain the finger quality value.
- uint16_t getCaptureEquipmentID () const

  Obtain the capture equipment identifier.
- bool isAppendixFCompliant () const

  Obtain the capture equipment compliance indicator for 'Appendix F'.
- std::shared_ptr< Image::Image > **getImage** () const

Static Public Member Functions

- static Finger::Position convertPosition (int incitsFGP)

  Convert a finger postion code from an INCITS finger record to the common code.
- static Finger::Impression convertImpression (int incitsIMP)

  Convert a impression type code from an INCITS finger record to the common code.

Protected Member Functions

- INCITSView (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view↩ Number)

  Construct the common components of an INCITS finger view from records contained in files.
- INCITSView (const Memory::uint8Array &fmrBuffer, const Memory::uint8Array &firBuffer, const uint32↩ _t viewNumber)

  Construct an INCITS finger view from records contained in buffers.
- Memory::uint8Array const & getFMRData () const

  Obtain a reference to the finger minutiae record data buffer.
- Memory::uint8Array const & getFIRData () const

  Obtain a reference to the finger image record data buffer.
- void setMinutiaeData (const Feature::INCITSMinutiae &fmd)

  Mutator for the Feature::INCITSMinutiae item.
- void setPosition (const Finger::Position &position)

  Mutator for the position.
- void setImpressionType (const Finger::Impression &impression)

  Mutator for the impression type.
- void setQuality (uint32_t quality)

  Mutator for the finger quality value.
- void setViewNumber (uint32_t viewNumber)

  Mutator for the finger view number.
- void setCaptureEquipmentID (uint16_t id)

  Mutator for the equipment ID.

- void setCBEFFProductIDs (uint16_t owner, uint16_t type)

  Mutator for the CBEFF Product ID owner and type.
- void setAppendixFCompliance (bool flag)

  Mutator for the Appendix F compliance indicator.
- void readFMRHeader (Memory::IndexedBuffer &buf, const uint32_t formatStandard)

  Read the common finger minutiae record header from an INCITS record.
- void readFVMR (Memory::IndexedBuffer &buf)

  Read the common finger view record information from an INCITS record.
- virtual Feature::MinutiaPointSet readMinutiaeDataPoints (Memory::IndexedBuffer &buf, uint32_t count)

  Read the minutiae data points, and extended data blocks.
- virtual void readExtendedDataBlock (Memory::IndexedBuffer &buf)

  Read the common extended data block.
- virtual Feature::RidgeCountItemSet readRidgeCountData (Memory::IndexedBuffer &buf, uint32_t dataLength)

  Read the ridge count data.
- virtual void readCoreDeltaData (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)=0

  Read the core points data.

Static Protected Attributes

- static const uint32_t **FMR_BASE_FORMAT_ID** = 0x464D5200
- static const uint32_t ANSI2004_STANDARD = 1

  The type of record that will be read by the subclass.
- static const uint32_t **ISO2005_STANDARD** = 2
- static const uint32_t **ANSI2007_STANDARD** = 3

H.53.1 Detailed Description

A class to represent single finger view and derived information.

A base Finger::INCITSView object represents an INCITS/ANSI or ISO finger view. This class defines the common interface for all ANSI/ISO views as well as common implementations. Subclasses specialize this class in order to represent other versions of the ANSI/ISO specs. Objects of this class cannot be created.

H.53.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::INCITSView::INCITSView (const std::string & *fmrFilename*, const std::string & *firFilename*, const uint32_t *viewNumber*) [protected]

Construct the common components of an INCITS finger view from records contained in files.

See documentation in child classes of INCITS for information on constructing INCITS-derived finger views.

Parameters

in	*fmrFilename*	The name of the file containing the complete finger minutiae record.

in	*firFilename*	The name of the file containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.
Error::FileError	Could not open or read from file.

BiometricEvaluation::Finger::INCITSView::INCITSView (const Memory::uint8Array & *fmrBuffer*, const Memory::uint8Array & *firBuffer*, const uint32_t *viewNumber*) [protected]

Construct an INCITS finger view from records contained in buffers.

See documentation in child classes of INCITS for information on constructing INCITS-derived finger views.

Parameters

in	*fmrBuffer*	The buffer containing the complete finger minutiae record.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.

H.53.3 Member Function Documentation

static Finger::Impression BiometricEvaluation::Finger::INCITSView::convertImpression (int *incitsIMP*) [static]

Convert a impression type code from an INCITS finger record to the common code.

Parameters

in	*incitsIMP*	A finger impression type code as defined by the INCITS standard.

Exceptions

Error::DataError	The impression type code is invalid.

Returns

The finger impression type code in common notation.

static Finger::Position BiometricEvaluation::Finger::INCITSView::convertPosition (int *incitsFGP*) [static]

Convert a finger postion code from an INCITS finger record to the common code.

Parameters

in	*incitsFGP*	A finger position code as defined by the INCITS standard.

Exceptions

Error::DataError	The position code is invalid.

Returns

The finger position code in common notation.

uint16_t BiometricEvaluation::Finger::INCITSView::getCaptureEquipmentID ( ) const

Obtain the capture equipment identifier.

Returns

The equipment ID.

Memory::uint8Array const& BiometricEvaluation::Finger::INCITSView::getFIRData ( ) const [protected]

Obtain a reference to the finger image record data buffer.

Returns

The entire finger image record data.

Memory::uint8Array const& BiometricEvaluation::Finger::INCITSView::getFMRData ( ) const [protected]

Obtain a reference to the finger minutiae record data buffer.

Returns

The entire finger minutiae record data.

Finger::Impression BiometricEvaluation::Finger::INCITSView::getImpressionType ( ) const

Obtain the finger impression code.

Returns

The finger impression code.

Finger::Position BiometricEvaluation::Finger::INCITSView::getPosition ( ) const

Obtain the finger position.

Returns

The finger position.

uint32_t BiometricEvaluation::Finger::INCITSView::getQuality ( ) const

Obtain the finger quality value.

Returns

The finger quality value.

bool BiometricEvaluation::Finger::INCITSView::isAppendixFCompliant ( ) const

Obtain the capture equipment compliance indicator for 'Appendix F'.

Returns

True if 'Appendix F' compliant, false otherwise.

virtual void BiometricEvaluation::Finger::INCITSView::readCoreDeltaData (Memory::Indexed↩ Buffer & *buf,* uint32_t *dataLength,* Feature::CorePointSet & *cores,* Feature::DeltaPointSet & *deltas*) [protected], [pure virtual]

Read the core points data.

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

Parameters

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

Implemented in BiometricEvaluation::Finger::ANSI2007View, BiometricEvaluation::Finger::ISO2005↩ View, and BiometricEvaluation::Finger::ANSI2004View.

virtual void BiometricEvaluation::Finger::INCITSView::readExtendedDataBlock (Memory::IndexedBuffer & *buf*) [protected], [virtual]

Read the common extended data block.

Parameters

in,out	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the extended data block.

Exceptions

DataError	The INCITS record has invalid or missing data.

void BiometricEvaluation::Finger::INCITSView::readFMRHeader (Memory::IndexedBuffer & *buf,* const uint32_t *formatStandard*) [protected]

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

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

Parameters

in	*buf*	The indexed buffer containing the record data. The index must start after the Format ID and spec version fields in the header. The index of the buffer will be changed to the location after the header.

in	*formatStandard*	Value indicating which header version to read; one of ANSI2004_STANDARD or ISO2005_STANDARD.

Exceptions

ParameterError	The specVersion parameter is incorrect.
DataError	The INCITS record has invalid or missing data.

void BiometricEvaluation::Finger::INCITSView::readFVMR (Memory::IndexedBuffer & *buf*) [protected]

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

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

Parameters

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

Exceptions

DataError	The INCITS record has invalid or missing data.

virtual Feature::MinutiaPointSet BiometricEvaluation::Finger::INCITSView::readMinutiaeDataPoints (Memory::IndexedBuffer & *buf,* uint32_t *count*) [protected], [virtual]

Read the minutiae data points, and extended data blocks.

Function to be implemented by derived classes to read the minutiae data points and extended data block according to the specifc standard they represent.

Parameters

in	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the finger view, including the extended data.
in	*count*	Number of minutiae data points to read.

Exceptions

DataError	The INCITS record has invalid or missing data.

virtual Feature::RidgeCountItemSet BiometricEvaluation::Finger::INCITSView::readRidgeCountData (Memory::IndexedBuffer & *buf,* uint32_t *dataLength*) [protected], [virtual]

Read the ridge count data.

This method reads data in the base INCITS format as defined in INCITS/ANSI 378-2004. This method may be overridden by derived classes to read data in a different record format.

Parameters

in,out	*buf*	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last ridge count item.
in	*dataLength*	The length of the entire ridge count data block.

void BiometricEvaluation::Finger::INCITSView::setAppendixFCompliance (bool *flag*) [protected]

Mutator for the Appendix F compliance indicator.
Parameters

in	*flag*	True if the capture equipment is 'Appendix F' compliant, false if not.

void BiometricEvaluation::Finger::INCITSView::setCaptureEquipmentID (uint16_t *id*) [protected]

Mutator for the equipment ID.
Parameters

in	*id*	The equipment ID value.

void BiometricEvaluation::Finger::INCITSView::setCBEFFProductIDs (uint16_t *owner*, uint16_t *type*) [protected]

Mutator for the CBEFF Product ID owner and type.
Parameters

in	*owner*	The CBEFF ID of the product owner.
in	*type*	The CBEFF ID of the product type.

void BiometricEvaluation::Finger::INCITSView::setImpressionType (const Finger::Impression & *impression*) [protected]

Mutator for the impression type.
Parameters

in	*impression*	The finger impression type code.

void BiometricEvaluation::Finger::INCITSView::setMinutiaeData (const Feature::INCITSMinutiae & *fmd*) [protected]

Mutator for the Feature::INCITSMinutiae item.
Parameters

in	*fmd*	The minutiae data object.

void BiometricEvaluation::Finger::INCITSView::setPosition (const Finger::Position & *position*) [protected]

Mutator for the position.

Parameters

in	*position*	The finger position.

void BiometricEvaluation::Finger::INCITSView::setQuality (uint32_t *quality*) [protected]

Mutator for the finger quality value.
Parameters

in	*quality*	The quality value.

void BiometricEvaluation::Finger::INCITSView::setViewNumber (uint32_t *viewNumber*) [protected]

Mutator for the finger view number.
Parameters

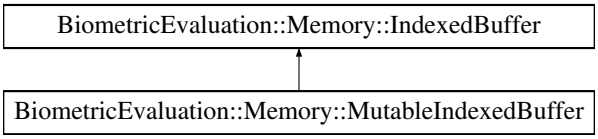
in	*viewNumber*	The view number value.

H.54 BiometricEvaluation::Memory::IndexedBuffer Class Reference

Wrap a memory buffer with an index.

`#include <be_memory_indexedbuffer.h>`

Inheritance diagram for BiometricEvaluation::Memory::IndexedBuffer:



Public Member Functions

- IndexedBuffer ()
- IndexedBuffer (const uint8_t *data, uint64_t size)

  Wrap an existing buffer of a given length.
- IndexedBuffer (const uint8Array &aa)

  Wrap an existing uint8Array.
- IndexedBuffer (const IndexedBuffer &copy)=default
- uint32_t getSize () const

  Obtain the current size of the buffer.
- uint32_t getIndex () const

  Obtain the current index into the buffer.
- void setIndex (uint64_t index)

  Set the current index into the buffer.
- uint8_t scanU8Val ()

  Obtain the next element of the buffer and increment the current index value.
- uint16_t scanU16Val ()

Obtain the next two elements of the buffer and increment the current index value.

- uint16_t scanBeU16Val ()

  Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.

- uint32_t scanU32Val ()

  Obtain the next four elements of the buffer and increment the current index value by four.

- uint32_t scanBeU32Val ()

  Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.

- uint64_t scanU64Val ()

  Obtain the next eight elements of the buffer and increment the current index value by eight.

- uint64_t scan (void *buf, uint64_t len)

  Obtain the next 'n' elements of the buffer and increment the current index value by n.

- virtual const uint8_t * get () const

  Returns a pointer to the managed buffer.

- virtual ~IndexedBuffer ()=default

H.54.1 Detailed Description

Wrap a memory buffer with an index.

The memory buffer is treated as an array of unsigned eight bit values. This class provides safe access to the array with methods to retrieve 8/16/32/64-bit elements, or and arbitrary segment starting at the index, from the array while advancing the current index. An exception is thrown by these methods whenever the retrieval would reach beyond the size of the buffer. IndexedBuffers do not own the memory of the buffers they wrap.

H.54.2 Constructor & Destructor Documentation

BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer ( )

Wrap a nullptr buffer.

BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (const uint8_t * *data*, uint64_t *size*)

Wrap an existing buffer of a given length.

Parameters

data	Buffer to wrap.
size	Size of buffer.

BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (const uint8Array & *aa*)

Wrap an existing uint8Array.

Parameters

aa	uint8Array to wrap.

BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (const IndexedBuffer & *copy*) `[default]`

Copy constructor (default).

virtual BiometricEvaluation::Memory::IndexedBuffer::~IndexedBuffer ( ) `[virtual], [default]`

Destructor (default).

H.54.3 Member Function Documentation

virtual const uint8_t* BiometricEvaluation::Memory::IndexedBuffer::get ( ) const `[virtual]`

Returns a pointer to the managed buffer.

Returns

Pointer to the managed buffer.

Reimplemented in BiometricEvaluation::Memory::MutableIndexedBuffer.

uint32_t BiometricEvaluation::Memory::IndexedBuffer::getIndex ( ) const

Obtain the current index into the buffer.

Returns

The current buffer index.

Note

When getIndex() == getSize(), the buffer is exhausted from scanning.

uint32_t BiometricEvaluation::Memory::IndexedBuffer::getSize ( ) const

Obtain the current size of the buffer.

Returns

The current buffer size.

uint64_t BiometricEvaluation::Memory::IndexedBuffer::scan (void * *buf*, uint64_t *len*)

Obtain the next 'n' elements of the buffer and increment the current index value by n.

Parameters

in	*buf*	Buffer to store the copied data, or nullptr.
in	*len*	The number of elements to copy.

Exceptions

Error::DataError	The buffer is exhausted.

Returns

The number of elements copied.

uint16_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU16Val ( )

Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.

Returns

The next element of the buffer as an unsigned 16-bit value.

Exceptions

Error::DataError	The buffer is exhausted.

uint32_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU32Val ( )

Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.

Returns

The next element of the buffer as an unsigned 32-bit value.

Exceptions

Error::DataError	The buffer is exhausted.

uint16_t BiometricEvaluation::Memory::IndexedBuffer::scanU16Val ( )

Obtain the next two elements of the buffer and increment the current index value.

Returns

The next element of the buffer as an unsigned 16-bit value.

Exceptions

Error::DataError	The buffer is exhausted.

uint32_t BiometricEvaluation::Memory::IndexedBuffer::scanU32Val ( )

Obtain the next four elements of the buffer and increment the current index value by four.

Returns

The next element of the buffer as an unsigned 32-bit value.

Exceptions

Error::DataError	The buffer is exhausted.

uint64_t BiometricEvaluation::Memory::IndexedBuffer::scanU64Val ( )

Obtain the next eight elements of the buffer and increment the current index value by eight.

Returns

The next element of the buffer as an unsigned 64-bit value.

Exceptions

Error::DataError	The buffer is exhausted.

uint8_t BiometricEvaluation::Memory::IndexedBuffer::scanU8Val ( )

Obtain the next element of the buffer and increment the current index value.

Returns

The next element of the buffer as an unsigned 8-bit value.

Exceptions

Error::DataError	The buffer is exhausted.

void BiometricEvaluation::Memory::IndexedBuffer::setIndex (uint64_t *index*)

Set the current index into the buffer.
Parameters

in	*index*	The index value to set.

Exceptions

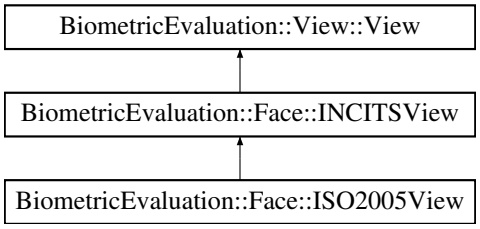
Error::ParameterError	The index parameter is too large.

H.55 BiometricEvaluation::Face::ISO2005View Class Reference

A class to represent single face view and derived information.

`#include <be_face_iso2005view.h>`

Inheritance diagram for BiometricEvaluation::Face::ISO2005View:



Public Member Functions

- ISO2005View ()

  Construct an empty ISO2005 Face Image Data record.
- ISO2005View (const std::string &filename, const uint32_t viewNumber)

  Construct an ISO 2005 face view from the named file.
- ISO2005View (const Memory::uint8Array &buffer, const uint32_t viewNumber)

  Construct an ISO 2005 face view from a record contained in a buffer.

Protected Member Functions

- void readISOHeader (BiometricEvaluation::Memory::IndexedBuffer &buf)

  Read the face image data record header from an ISO 2005 record.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x30313000

H.55.1 Detailed Description

A class to represent single face view and derived information.

A base Face::ISO2005View class represents an ISO 2005 face image data view.

H.55.2 Constructor & Destructor Documentation

BiometricEvaluation::Face::ISO2005View::ISO2005View (const std::string & *filename*, const uint32_t *viewNumber*)

Construct an ISO 2005 face view from the named file.

The entire face image data record is passed into this method, with the specific instance of the facial image that is to be extraced from the record.

Parameters

in	*filename*	The name of the file containing the complete face image data record.
in	*viewNumber*	The facial information instance to read.

Exceptions

Error::DataError	Invalid record format.
Error::FileError	Could not open or read from file.

BiometricEvaluation::Face::ISO2005View::ISO2005View (const Memory::uint8Array & *buffer*, const uint32_t *viewNumber*)

Construct an ISO 2005 face view from a record contained in a buffer.

The entire face image data record is passed into this method, with the specific instance of the facial image that is to be extraced from the record.

Parameters

in	*buffer*	The buffer containing the complete face image data record.
in	*viewNumber*	The facial information instance to read.

Exceptions

Error::DataError	Invalid record format.

H.55.3 Member Function Documentation

void BiometricEvaluation::Face::ISO2005View::readISOHeader (BiometricEvaluation::Memory::↩ IndexedBuffer & *buf*) [protected]

Read the face image data record header from an ISO 2005 record.

Parameters

in	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the header.

Exceptions

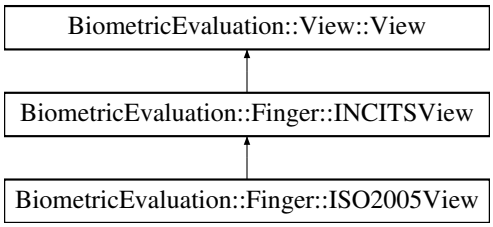
DataError	The record has invalid or missing data.

H.56 BiometricEvaluation::Finger::ISO2005View Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_iso2005view.h>`

Inheritance diagram for BiometricEvaluation::Finger::ISO2005View:



Public Member Functions

- ISO2005View (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber)

  Construct an ISO-2005 finger view from records contained in files.
- ISO2005View (Memory::uint8Array &fmrBuffer, Memory::uint8Array &firBuffer, const uint32_t viewNumber)

  Construct an ISO-2005 finger view from records contained in buffers.

Protected Member Functions

- void **readFMRHeader** (Memory::IndexedBuffer &buf)
- void readCoreDeltaData (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)

  Read the core points data.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x20323000

Additional Inherited Members

H.56.1 Detailed Description

A class to represent single finger view and derived information.

A Finger::ISO2005View object represents a finger view from a ISO/IEC-2005 Finger Minutiae Record.

H.56.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::ISO2005View::ISO2005View (const std::string & *fmrFilename,* const std::string & *firFilename,* const uint32_t *viewNumber*)

Construct an ISO-2005 finger view from records contained in files.

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

Parameters

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

BiometricEvaluation::Finger::ISO2005View::ISO2005View (Memory::uint8Array & *fmrBuffer,* Memory::uint8Array & *firBuffer,* const uint32_t *viewNumber*)

Construct an ISO-2005 finger view from records contained in buffers.

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

Parameters

in	*fmrBuffer*	The buffer containing the complete finger minutiae record.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.

H.56.3 Member Function Documentation

void BiometricEvaluation::Finger::ISO2005View::readCoreDeltaData (Memory::IndexedBuffer & *buf,* uint32_t *dataLength,* Feature::CorePointSet & *cores,* Feature::DeltaPointSet & *deltas*) [protected], [virtual]

Read the core points data.

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

Parameters

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

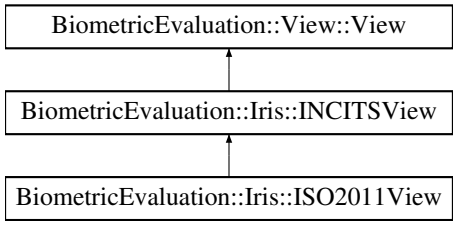
Implements BiometricEvaluation::Finger::INCITSView.

H.57 BiometricEvaluation::Iris::ISO2011View Class Reference

A class to represent single iris view and derived information.

`#include <be_iris_iso2011view.h>`

Inheritance diagram for BiometricEvaluation::Iris::ISO2011View:



Public Member Functions

- ISO2011View ()

  Construct an empty ISO 2011 iris view.
- ISO2011View (const std::string &filename, const uint32_t viewNumber)

  Construct an ISO 2011 iris view from the named file.
- ISO2011View (const Memory::uint8Array &buffer, const uint32_t viewNumber)

  Construct an ISO 2011 iris view from a record contained in a buffer.

Protected Member Functions

- void **readISOHeader** (BiometricEvaluation::Memory::IndexedBuffer &buf)

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x30323000

Additional Inherited Members

H.57.1 Detailed Description

A class to represent single iris view and derived information.

An Iris::ISO2011VIEW class represents an ISO 19794-6 iris image record view.

H.57.2 Constructor & Destructor Documentation

BiometricEvaluation::Iris::ISO2011View::ISO2011View (const std::string & *filename*, const uint32_t *viewNumber*)

Construct an ISO 2011 iris view from the named file.

Parameters

in	*filename*	The name of the file containing the complete iris image record.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError	Invalid record format.
Error::FileError	Could not open or read from file.

BiometricEvaluation::Iris::ISO2011View::ISO2011View (const Memory::uint8Array & *buffer*, const uint32_t *viewNumber*)

Construct an ISO 2011 iris view from a record contained in a buffer.

Parameters

in	*buffer*	The buffer containing the complete iris image record.
in	*viewNumber*	The eye number to use.

Exceptions

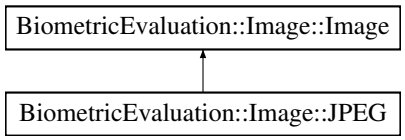
Error::DataError	Invalid record format.

H.58 BiometricEvaluation::Image::JPEG Class Reference

A JPEG-encoded image.

`#include <be_image_jpeg.h>`

Inheritance diagram for BiometricEvaluation::Image::JPEG:



Public Member Functions

- **JPEG** (const uint8_t *data, const uint64_t size)
- Memory::uint8Array getRawGrayscaleData (uint8_t depth=8) const

  Accessor for decompressed data in grayscale.
- Memory::uint8Array getRawData () const

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

Static Public Member Functions

- static bool isJPEG (const uint8_t *data, uint64_t size)
- static int **getc_skip_marker_segment** (const unsigned short marker, unsigned char **cbufptr, unsigned char *ebufptr)

Additional Inherited Members

H.58.1 Detailed Description

A JPEG-encoded image.

H.58.2 Member Function Documentation

Memory::uint8Array BiometricEvaluation::Image::JPEG::getRawData ( ) const [virtual]

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

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError	Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

Memory::uint8Array BiometricEvaluation::Image::JPEG::getRawGrayscaleData (uint8_t *depth* = *8*) const [virtual]

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

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

Note

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

Implements BiometricEvaluation::Image::Image.

static bool BiometricEvaluation::Image::JPEG::isJPEG (const uint8_t * *data*, uint64_t *size*) [static]

Whether or not data is a Lossy JPEG image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

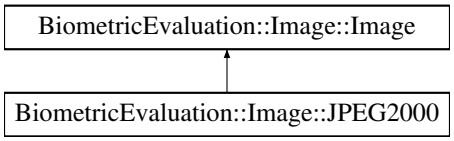
true if data appears to be a Lossy JPEG image, false otherwise

H.59 BiometricEvaluation::Image::JPEG2000 Class Reference

A JPEG-2000-encoded image.

`#include <be_image_jpeg2000.h>`

Inheritance diagram for BiometricEvaluation::Image::JPEG2000:



Public Member Functions

- JPEG2000 (const uint8_t *data, const uint64_t size, const int8_t codec=2)

  Create a new JPEG2000 object.
- Memory::uint8Array getRawData () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- Memory::uint8Array getRawGrayscaleData (uint8_t depth=8) const

  Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isJPEG2000 (const uint8_t *data, uint64_t size)

Additional Inherited Members

H.59.1 Detailed Description

A JPEG-2000-encoded image.

H.59.2 Constructor & Destructor Documentation

BiometricEvaluation::Image::JPEG2000::JPEG2000 (const uint8_t * *data*, const uint64_t *size*, const int8_t *codec = 2*)

Create a new JPEG2000 object.

Parameters

in	*data*	The image data.
in	*size*	The size of the image data, in bytes.
in	*codec*	The OPJ_CODEC_FORMAT used to encode data.

Exceptions

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

H.59.3 Member Function Documentation

Memory::uint8Array BiometricEvaluation::Image::JPEG2000::getRawData ( ) const [virtual]

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError	Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

Memory::uint8Array BiometricEvaluation::Image::JPEG2000::getRawGrayscaleData (uint8_t *depth* = *8*) const [virtual]

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 8 or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError	Error decompressing image data.
Error::ParameterError	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

static bool BiometricEvaluation::Image::JPEG2000::isJPEG2000 (const uint8_t * *data*, uint64_t *size*) [static]

Whether or not data is a JPEG-2000 image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

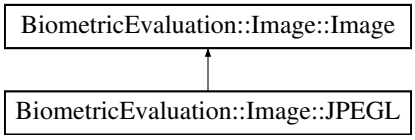
true if data appears to be a JPEG-2000 image, false otherwise.

H.60 BiometricEvaluation::Image::JPEGL Class Reference

A Lossless JPEG-encoded image.

`#include <be_image_jpegl.h>`

Inheritance diagram for BiometricEvaluation::Image::JPEGL:



Public Member Functions

- **JPEGL** (const uint8_t *data, const uint64_t size)
- Memory::uint8Array getRawGrayscaleData (uint8_t depth=8) const

  Accessor for decompressed data in grayscale.
- Memory::uint8Array getRawData () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.

Static Public Member Functions

- static bool isJPEGL (const uint8_t *data, uint64_t size)

Additional Inherited Members

H.60.1 Detailed Description

A Lossless JPEG-encoded image.

H.60.2 Member Function Documentation

Memory::uint8Array BiometricEvaluation::Image::JPEGL::getRawData ( ) const `[virtual]`

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError	Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

Memory::uint8Array BiometricEvaluation::Image::JPEGL::getRawGrayscaleData (uint8_t *depth = 8*) const `[virtual]`

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 8 or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError	Error decompressing image data.
Error::ParameterError	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

static bool BiometricEvaluation::Image::JPEGL::isJPEGL (const uint8_t * *data*, uint64_t *size*) [static]

Whether or not data is a Lossless JPEG image.
Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

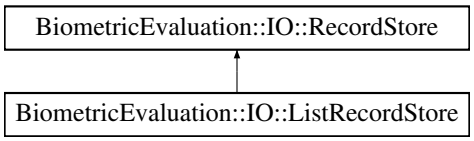
true if data appears to be a Lossless JPEG image, false otherwise.

H.61 BiometricEvaluation::IO::ListRecordStore Class Reference

RecordStore that reads a list of keys from a text file, and retrieves the data from another RecordStore.

`#include <be_io_listrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ListRecordStore:



Public Member Functions

- ListRecordStore (const std::string &pathname)
- ~ListRecordStore ()
- void insert (const std::string &key, const void *const data, const uint64_t size)

- void remove (const std::string &key)
- uint64_t read (const std::string &key, void *const data) const
- void replace (const std::string &key, const void *const data, const uint64_t size)
- uint64_t length (const std::string &key) const
- void flush (const std::string &key) const
- void sync () const
- uint64_t sequence (std::string &key, void *const data=nullptr, int cursor=BE_RECSTORE_SEQ_NEXT)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (const std::string &key)
- void move (const std::string &pathname)

  Move the RecordStore.
- uint64_t getSpaceUsed () const

  Obtain real storage utilization.

Static Public Attributes

- static const std::string SOURCERECORDSTOREPROPERTY
- static const std::string KEYLISTFILENAME

Additional Inherited Members

H.61.1 Detailed Description

RecordStore that reads a list of keys from a text file, and retrieves the data from another RecordStore.

ListRecordStores must be hand-crafted by first setting the 'Source Record Store', 'Type', and 'Count' properties in the .rscontrol.prop file. 'Source Record Store' is the complete path of the RecordStore containing the actual data records. Type must be 'List'. Count should match the number of entries in the file created next. Other properties are as in a "normal" RecordStore; see example below.

Second, create a file called 'KeyList.txt' in the RecordStore directory containing a list of keys, one per line.

ListRecordStores can also be created and modified with versions of rstool(1) from 2013 or later.

Example .rscontrol.prop file: Count = 10 Description = Search records for SDK TESTSDK Name = Test↩ LRS Type = List Source Record Store = /Users/wsalamon/sandbox/SD29.rs

Note

List RecordStores must be opened read-only.

H.61.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::ListRecordStore::ListRecordStore (const std::string & *pathname*)

Constructor, always opening read-only

BiometricEvaluation::IO::ListRecordStore::~ListRecordStore ( )

Destructor

H.61.3 Member Function Documentation

void BiometricEvaluation::IO::ListRecordStore::flush (const std::string & *key*) const [virtual]

Commit the record's data to storage.

Parameters

in	*key*	The key of the record to be flushed.

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::ListRecordStore::getSpaceUsed ( ) const [virtual]

Obtain real storage utilization.

The amount of disk space used, for example. This is the actual space allocated by the underlying storage mechanism, in bytes.

Returns

The amount of backing storage used by the RecordStore.

Exceptions

Error::StrategyError	An error occurred when using the underlying storage system.

Reimplemented from BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ListRecordStore::insert (const std::string & *key*, const void *const *data*, const uint64_t *size*) [virtual]

Insert a record into the store.

Parameters

in	*key*	The key of the record to be inserted.
in	*data*	The data for the record.
in	*size*	The size, in bytes, of the record.

Exceptions

Error::ObjectExists	A record with the given key is already present.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::ListRecordStore::length (const std::string & *key*) const [virtual]

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ListRecordStore::move (const std::string & *pathname*) [virtual]

Move the RecordStore.

The RecordStore can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the RecordStore.

Exceptions

Error::StrategyError	An error occurred when using the underlying storage system.

Reimplemented from BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::ListRecordStore::read (const std::string & *key*, void *const *const *data*) const [virtual]

Read a complete record from a store. Applications are responsible for allocating storage for the record's data.

Parameters

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ListRecordStore::remove (const std::string & *key*) [virtual]

Remove a record from the store.

Parameters

in	*key*	The key of the record to be removed.

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ListRecordStore::replace (const std::string & *key,* const void *const *data,* const uint64_t *size*) [virtual]

Replace a complete record in a store.

Parameters

in	*key*	The key of the record to be replaced.
in	*data*	The data for the record.
in	*size*	The size of data.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::ListRecordStore::sequence (std::string & *key*, void *const *data = *nullptr*, int *cursor* = BE_RECSTORE_SEQ_NEXT) [virtual]

Sequence through a RecordStore, returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the sequencor to return the next record. The starting point is typically the first record, and is set to that when the RecordStore object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to nullptr to indicate only the key is wanted.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ListRecordStore::setCursorAtKey (const std::string & *key*) [virtual]

Set the sequence cursor to an arbitrary position within the RecordStore, starting at key. Key will be the first record returned from the next call to sequence().

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to sequence().

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ListRecordStore::sync ( ) const `[virtual]`

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError	An error occurred when using the underlying storage system.

Reimplemented from BiometricEvaluation::IO::RecordStore.

H.61.4 Member Data Documentation

const std::string BiometricEvaluation::IO::ListRecordStore::KEYLISTFILENAME [static]

File name containing the list of keys

const std::string BiometricEvaluation::IO::ListRecordStore::SOURCERECORDSTOREPROPERTY [static]

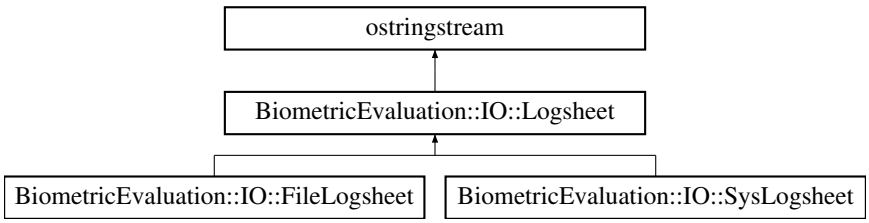
Property key for the source RecordStore

H.62 BiometricEvaluation::IO::Logsheet Class Reference

A class to represent a logging mechanism.

`#include <be_io_logsheet.h>`

Inheritance diagram for BiometricEvaluation::IO::Logsheet:



Public Types

- enum Kind { Kind::Null, Kind::File, Kind::Syslog }

Public Member Functions

- Logsheet ()

  Create a Logsheet that has no backing store. A log entry is maintained, but cannot be permanently stored. This is the Null Logsheet.
- virtual ~Logsheet ()
- void newEntry ()

  Start a new entry, causing the existing entry to be closed and written.
- std::string getCurrentEntry () const

  Obtain the contents of the current entry currently under construction.
- void resetCurrentEntry ()
- uint32_t getCurrentEntryNumber () const

  Obtain the current entry number.
- virtual void write (const std::string &entry)

  Write a string as an entry to the backing store.

- virtual void writeComment (const std::string &entry)

  Write a string as a comment to the backing store.
- virtual void writeDebug (const std::string &entry)

  Write a string as a debug entry to the backing store.
- void setCommit (const bool state)

  Enable or disable the commitment of normal entries to the backing log storage.
- bool getCommit () const

  Get the current entry commit state.
- void setDebugCommit (const bool state)

  Enable or disable the commitment of debug entries to the backing log storage.
- bool getDebugCommit () const

  Get the current debug entry commit state.
- void setCommentCommit (const bool state)

  Enable or disable the commitment of comment entries to the backing log storage.
- bool getCommentCommit () const

  Get the current comment entry commit state.
- virtual void sync ()

  Synchronize any buffered data to the underlying backing store.
- void setAutoSync (bool state)
- bool getAutoSync () const

Static Public Member Functions

- static Logsheet::Kind getTypeFromURL (const std::string &url)

  Map the URL scheme, taken from a string containing the entire URL, into a Logsheet type.
- static bool lineIsEntry (const std::string &line)

  Helper function to determine whether a string is a valid log entry.
- static bool lineIsComment (const std::string &line)

  Helper function to determine whether a string is a valid comment log entry.
- static bool lineIsDebug (const std::string &line)

  Helper function to determine whether a string is a valid debug log entry.
- static std::string trim (const std::string &entry)

  Trim delimiters from Logsheet entries.

Static Public Attributes

- static const char CommentDelimiter = '#'
- static const char EntryDelimiter = 'E'
- static const char DebugDelimiter = 'D'
- static const std::string DescriptionTag
- static const std::string FILEURLSCHEME
- static const std::string SYSLOGURLSCHEME

Protected Member Functions

- void incrementEntryNumber ()

  Increment the current entry number.

- std::string getCurrentEntryNumberAsString () const

  Obtain the current entry 'tag', in 'Edddd' format.

H.62.1 Detailed Description

A class to represent a logging mechanism.

A Logsheet is a string stream, so applications can write into the stream as a staging area using the << operator, then start a new entry by calling newEntry(). Entries in the log are prefixed with an entry number, which is incremented when the entry is written (either by directly calling write(), or calling newEntry()).

How the log data is stored is implemented by subclasses of Logsheet.

Note

By default, the entries in the Logsheet may not be immediately written to the backing store, depending on the buffering behavior of the operating system. Applications can force a write by invoking sync(), or force a write at every new log entry by invoking setAutoSync(true).
Entries created by applications may be composed of more than one line (each separated by the newline character). The text at the beginning of a line should not "look like" an entry number:
Edddd
i.e. the entry delimiter followed by some digits. Logsheet won't check for that condition, but any existing Logsheet that is re-opened for append may have an incorrect starting entry number.

H.62.2 Member Enumeration Documentation

enum BiometricEvaluation::IO::Logsheet::Kind `[strong]`

Enumerator

Null No backing store log sheet

File File-based log sheet

Syslog Syslog daemon backing store

H.62.3 Constructor & Destructor Documentation

virtual BiometricEvaluation::IO::Logsheet::~Logsheet ( ) `[virtual]`

Destructor

H.62.4 Member Function Documentation

bool BiometricEvaluation::IO::Logsheet::getAutoSync ( ) const

Return the current auto-sync state.

Returns

true if auto-sync is on, false otherwise.

bool BiometricEvaluation::IO::Logsheet::getCommentCommit ( ) const

Get the current comment entry commit state.

Returns

true if comment entries are committed to the backing store, false otherwise.

bool BiometricEvaluation::IO::Logsheet::getCommit ( ) const

Get the current entry commit state.

Returns

true if normal entries are to be committed, false if not.

std::string BiometricEvaluation::IO::Logsheet::getCurrentEntry ( ) const

Obtain the contents of the current entry currently under construction.

Returns

The text of the current entry.

uint32_t BiometricEvaluation::IO::Logsheet::getCurrentEntryNumber ( ) const

Obtain the current entry number.

Returns

The current entry number.

std::string BiometricEvaluation::IO::Logsheet::getCurrentEntryNumberAsString ( ) const `[protected]`

Obtain the current entry 'tag', in 'Edddd' format.

Returns

The text of the current entry tag.

bool BiometricEvaluation::IO::Logsheet::getDebugCommit ( ) const

Get the current debug entry commit state.

Returns

true if debug entries are committed to the backing store, false otherwise.

static Logsheet::Kind BiometricEvaluation::IO::Logsheet::getTypeFromURL (const std::string & *url*) `[static]`

Map the URL scheme, taken from a string containing the entire URL, into a Logsheet type.

Parameters

in	*url*	The unform resource locator of the Logsheet.

Returns

The type of Logsheet represented by the URL.

Exceptions

Error::ParameterError	The URL scheme is missing or invalid.

static bool BiometricEvaluation::IO::Logsheet::lineIsComment (const std::string & *line*) [static]

Helper function to determine whether a string is a valid comment log entry.
Parameters

in	*line*	The string potentially containing a comment entry.

Returns

true if the string is a comment entry, false otherwise.

static bool BiometricEvaluation::IO::Logsheet::lineIsDebug (const std::string & *line*) [static]

Helper function to determine whether a string is a valid debug log entry.
Parameters

in	*line*	The string potentially containing a debug entry.

Returns

true if the string is a debug entry, false otherwise.

static bool BiometricEvaluation::IO::Logsheet::lineIsEntry (const std::string & *line*) [static]

Helper function to determine whether a string is a valid log entry.
Parameters

in	*line*	The string potentially containing a log entry.

Returns

true if the string is a log entry, false otherwise.

void BiometricEvaluation::IO::Logsheet::newEntry ( )

Start a new entry, causing the existing entry to be closed and written.

Applications do not have to call this method for the first entry, however, as the stream is ready for writing upon construction.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

void BiometricEvaluation::IO::Logsheet::resetCurrentEntry ( )

Reset the current entry buffer to the beginning.

void BiometricEvaluation::IO::Logsheet::setAutoSync (bool *state*)

Turn on/off auto-sync of the data. Applications may gain performance by turning off auto-sync, or gain reliability by turning it on.

Parameters

state	When true, the data is sync'd whenever newEntry() is or write() is called. When false, sync() must be called to force a write.

void BiometricEvaluation::IO::Logsheet::setCommentCommit (const bool *state*)

Enable or disable the commitment of comment entries to the backing log storage.

When comment entry commitment is disabled, calls to writeComment may still be made, but those entries do not appear in the log backing store.

Parameters

in	*state*	true if comment entries are to be committed, false if not.

void BiometricEvaluation::IO::Logsheet::setCommit (const bool *state*)

Enable or disable the commitment of normal entries to the backing log storage.

When entry commitment is disabled, the entry number is not incremented. Entries may be streamed into the object, and new entries created.

Parameters

in	*state*	True if normal entries are to be committed, false if not.

void BiometricEvaluation::IO::Logsheet::setDebugCommit (const bool *state*)

Enable or disable the commitment of debug entries to the backing log storage.

When debug entry commitment is disabled, calls to writeDebug may still be made, but those entries do not appear in the log backing store.

Parameters

in	*state*	true if debug entries are to be committed, false if not.

virtual void BiometricEvaluation::IO::Logsheet::sync ( ) [virtual]

Synchronize any buffered data to the underlying backing store.

This syncing is dependent on the behavior of the underlying storage mechanism.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

Reimplemented in BiometricEvaluation::IO::FileLogsheet, and BiometricEvaluation::IO::SysLogsheet.

static std::string BiometricEvaluation::IO::Logsheet::trim (const std::string & *entry*) [static]

Trim delimiters from Logsheet entries.

Works for comments and numbered entries.

Parameters

in	*entry*	The entry to trim.

Returns

Delimiter-less entry.

virtual void BiometricEvaluation::IO::Logsheet::write (const std::string & *entry*) [virtual]

Write a string as an entry to the backing store.

This does not affect the current log entry buffer, but does increment the entry number.

Parameters

in	*entry*	The text of the log entry.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

Reimplemented in BiometricEvaluation::IO::FileLogsheet, and BiometricEvaluation::IO::SysLogsheet.

virtual void BiometricEvaluation::IO::Logsheet::writeComment (const std::string & *entry*) [virtual]

Write a string as a comment to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

in	*entry*	The text of the comment.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

Reimplemented in BiometricEvaluation::IO::FileLogsheet, and BiometricEvaluation::IO::SysLogsheet.

virtual void BiometricEvaluation::IO::Logsheet::writeDebug (const std::string & *entry*) [virtual]

Write a string as a debug entry to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A debug line is prefixed with DebugDelimiter followed by a space.

Parameters

in	*entry*	The text of the debug message.

Exceptions

Error::StrategyError	An error occurred when logging.

Reimplemented in BiometricEvaluation::IO::FileLogsheet, and BiometricEvaluation::IO::SysLogsheet.

H.62.5 Member Data Documentation

const char BiometricEvaluation::IO::Logsheet::CommentDelimiter = '#' [static]

Delimiter for a comment line in the log sheet.

const char BiometricEvaluation::IO::Logsheet::DebugDelimiter = 'D' [static]

Delimiter for an debug line in the log sheet.

const std::string BiometricEvaluation::IO::Logsheet::DescriptionTag [static]

The tag for the description string.

const char BiometricEvaluation::IO::Logsheet::EntryDelimiter = 'E' [static]

Delimiter for an entry line in the log sheet.

const std::string BiometricEvaluation::IO::Logsheet::FILEURLSCHEME [static]

The URL scheme to be used for FileLogsheet URL strings.

const std::string BiometricEvaluation::IO::Logsheet::SYSLOGURLSCHEME [static]

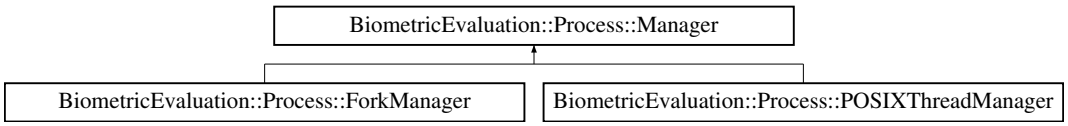
The URL scheme to be used for SysLogsheet URL strings.

H.63 BiometricEvaluation::Process::Manager Class Reference

An interface for intranode process management classes.

`#include <be_process_manager.h>`

Inheritance diagram for BiometricEvaluation::Process::Manager:



Public Member Functions

- Manager ()

  Manager constructor.
- virtual std::shared_ptr < WorkerController > addWorker (std::shared_ptr< Worker > worker)=0

Adds a Worker to be managed by this Manager.

- virtual uint32_t getNumCompletedWorkers () const

  Obtain the number of Workers that have exited.

- virtual uint32_t getNumActiveWorkers () const

  Obtain the number of Workers that are still working.

- virtual uint32_t getTotalWorkers () const

  Obtain the number of Workers this class is handling.

- virtual void startWorkers (bool wait=true, bool communicate=false)=0

  Begin Worker's work.

- virtual void startWorker (std::shared_ptr< WorkerController > worker, bool wait=true, bool communicate=false)=0

  Start a Worker.

- virtual void waitForWorkerExit ()=0

  Block until all Workers have exited.

- virtual void reset ()

  Reuse all Workers.

- virtual int32_t stopWorker (std::shared_ptr< WorkerController > worker)=0

  Ask Worker to return as soon as possible.

- virtual bool waitForMessage (std::shared_ptr< WorkerController > &sender, int *nextFD=nullptr, int numSeconds=-1) const

  Wait for a message from a Worker.

- virtual bool getNextMessage (std::shared_ptr< WorkerController > &sender, Memory::uint8Array &message, int numSeconds=-1) const

  Obtain a message from a Worker.

- virtual void broadcastMessage (Memory::uint8Array &message) const

  Send one message to all Workers.

- virtual ~Manager ()

  Manager destructor.

Protected Member Functions

- virtual void _wait ()=0

  Do not return until all spawned processes exited.

Protected Attributes

- std::vector< std::shared_ptr < WorkerController > > _workers
- std::vector< std::shared_ptr < WorkerController > > _pendingExit

H.63.1 Detailed Description

An interface for intranode process management classes.

H.63.2 Member Function Documentation

virtual std::shared_ptr<WorkerController> BiometricEvaluation::Process::Manager::addWorker (std::shared_ptr< Worker > *worker*) [pure virtual]

Adds a Worker to be managed by this Manager.

Parameters

worker	A Worker instance to run.

Returns

shared_ptr to worker.

Implemented in BiometricEvaluation::Process::ForkManager, and BiometricEvaluation::Process::POSIXThreadManager.

virtual void BiometricEvaluation::Process::Manager::broadcastMessage (Memory::uint8Array & *message*) const [virtual]

Send one message to all Workers.

Parameters

message	The message to send to all Workers.

Exceptions

Error::StrategyError	Error propagated from the WorkerController.

virtual bool BiometricEvaluation::Process::Manager::getNextMessage (std::shared_ptr< WorkerController > & *sender*, Memory::uint8Array & *message*, int *numSeconds* = *-1*) const [virtual]

Obtain a message from a Worker.

Parameters

out	*sender*	Reference to a shared pointer of the WorkerController that sent the message.
out	*message*	Reference to a buffer to hold the message.
in	*numSeconds*	Number of seconds to wait for a message, or < 0 to block.

Returns

true if there is a message, false otherwise.

Exceptions

Error::ObjectDoesNotExist	(Unexpected) widowed pipe.
Error::StrategyError	Error receiving message.

virtual uint32_t BiometricEvaluation::Process::Manager::getNumActiveWorkers ( ) const [virtual]

Obtain the number of Workers that are still working.

Returns

The number of Workers that are still working.

Exceptions

Error::StrategyError	No Workers have started working yet.

virtual uint32_t BiometricEvaluation::Process::Manager::getNumCompletedWorkers ( ) const [virtual]

Obtain the number of Workers that have exited.

Returns

The number of Workers that have exited.

Exceptions

Error::StrategyError	No Workers have started working yet.

virtual uint32_t BiometricEvaluation::Process::Manager::getTotalWorkers ( ) const [virtual]

Obtain the number of Workers this class is handling.

Returns

Number of Workers.

virtual void BiometricEvaluation::Process::Manager::reset ( ) [virtual]

Reuse all Workers.

Exceptions

Error::ObjectExists	At least one Worker is still working.

virtual void BiometricEvaluation::Process::Manager::startWorker (std::shared_ptr< WorkerController > *worker*, bool *wait* = true, bool *communicate* = false) [pure virtual]

Start a Worker.

Parameters

	worker	Pointer to a WorkerController that is being managed by this Manager instance.
	wait	Whether or not to wait for this Worker to exit before returning control to the caller.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

Note

Some implementations of this interface may call the system exit function from this routine. Therefore, the application's implementation of workerMain() should release all resources before returning.

Implemented in BiometricEvaluation::Process::ForkManager, and BiometricEvaluation::Process::POSIXThreadManager.

virtual void BiometricEvaluation::Process::Manager::startWorkers (bool *wait* = `true`, bool *communicate* = `false`) [pure virtual]

Begin Worker's work.

Parameters

in	*wait*	Whether or not to wait for all Workers to return before returning.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	At least one Worker is already working.
Error::StrategyError	Problem starting Workers.

Implemented in BiometricEvaluation::Process::ForkManager, and BiometricEvaluation::Process::POSIXThreadManager.

virtual int32_t BiometricEvaluation::Process::Manager::stopWorker (std::shared_ptr< WorkerController > *worker*) [pure virtual]

Ask Worker to return as soon as possible.

Parameters

worker	Pointer to the WorkerController that should be stopped.

Returns

Return code of worker.

Exceptions

Error::ObjectDoesNotExist	worker is not working.
Error::StrategyError	Problem asking worker to stop.

Implemented in BiometricEvaluation::Process::ForkManager, and BiometricEvaluation::Process::POSIXThreadManager.

virtual bool BiometricEvaluation::Process::Manager::waitForMessage (std::shared_ptr< WorkerController > & *sender*, int * *nextFD* = *nullptr*, int *numSeconds* = *-1*) const [virtual]

Wait for a message from a Worker.

Parameters

out	*sender*	Reference to a shared pointer of the WorkerController that sent the message.
in,out	*nextFD*	Location to store a pipe that has data to read.
in	*numSeconds*	Number of seconds to wait for a message, or < 0 to block.

Returns

true if there is a Worker sending a message false otherwise or if an error occurred.

virtual void BiometricEvaluation::Process::Manager::waitForWorkerExit ( ) [pure virtual]

Block until all Workers have exited.

Use this method if wait=false was set during a call to startWorker(s) but now wait=true is desired.

Implemented in BiometricEvaluation::Process::ForkManager, and BiometricEvaluation::Process::POSIXThreadManager.

H.63.3 Member Data Documentation

std::vector<std::shared_ptr<WorkerController> > BiometricEvaluation::Process::Manager::pendingExit [protected]

Workers that are about to exit (stop requested).

std::vector<std::shared_ptr<WorkerController> > BiometricEvaluation::Process::Manager::workers [protected]

Workers that have been added.

H.64 BiometricEvaluation::IO::ManifestEntry Struct Reference

`#include <be_io_archiverecstore.h>`

Public Attributes

- long offset
- uint64_t size

H.64.1 Detailed Description

Info about a single archive element

H.64.2 Member Data Documentation

long BiometricEvaluation::IO::ManifestEntry::offset

The offset from the beginning of the file/memory

uint64_t BiometricEvaluation::IO::ManifestEntry::size

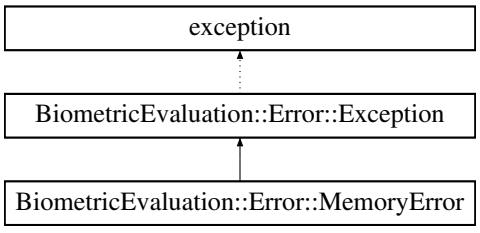
The length from offset this element spans

H.65 BiometricEvaluation::Error::MemoryError Class Reference

An error occurred when allocating an object.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::MemoryError:



Public Member Functions

- MemoryError ()
- MemoryError (std::string info)

H.65.1 Detailed Description

An error occurred when allocating an object.

H.65.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::MemoryError::MemoryError ( )

Construct a MemoryError object with the default information string.

BiometricEvaluation::Error::MemoryError::MemoryError (std::string *info*)

Construct a MemoryError object with an information string appended to the default information string.

H.66 BiometricEvaluation::Process::MessageCenter Class Reference

`#include <be_process_messagecenter.h>`

Public Member Functions

- MessageCenter (uint32_t port=MessageCenter::DEFAULT_PORT)

  Constructor.
- bool hasUnseenMessages () const

  Determine whether or not there are unseen messages.
- bool getNextMessage (uint32_t &clientID, Memory::uint8Array &message, int numSeconds=-1)

  Get the next available message.
- void sendResponse (uint32_t clientID, const Memory::uint8Array &message) const

  Send a message to a client.
- void disconnectClient (uint32_t clientID)

  Break the connection with a client.

Static Public Attributes

- static const int CONNECTION_BACKLOG = 10
- static const uint16_t DEFAULT_PORT = 7899
- static const int DEFAULT_TIMEOUT = 1
- static const uint64_t MAX_MESSAGE_LENGTH = 255

H.66.1 Detailed Description

Convenience for asynchronous TCP socket message passing.

H.66.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::MessageCenter::MessageCenter (uint32_t *port* = MessageCenter::DEFAULT_PORT)

Constructor.

Parameters

port	Listening port.

H.66.3 Member Function Documentation

void BiometricEvaluation::Process::MessageCenter::disconnectClient (uint32_t *clientID*)

Break the connection with a client.
Parameters

clientID	ID of the client to disconect.

bool BiometricEvaluation::Process::MessageCenter::getNextMessage (uint32_t & *clientID*, Memory::uint8Array & *message*, int *numSeconds* = *-1*)

Get the next available message.
Parameters

out	*clientID*	ID of the client that sent the message.
in,out	*message*	Message received.
in	*numSeconds*	Number of seconds to wait for a message, or < 0 to block indefinitely.

Returns

true if a message was received before timing out.

bool BiometricEvaluation::Process::MessageCenter::hasUnseenMessages ( ) const

Determine whether or not there are unseen messages.

Returns

true if a message has been received and not read.

Note

Returns immediately.

void BiometricEvaluation::Process::MessageCenter::sendResponse (uint32_t *clientID*, const Memory::uint8Array & *message*) const

Send a message to a client.
Parameters

clientID	ID of client to receive message.
message	Message to send client.

H.66.4 Member Data Documentation

const int BiometricEvaluation::Process::MessageCenter::CONNECTION_BACKLOG = 10 [static]

Number of outstanding connections.

const uint16_t BiometricEvaluation::Process::MessageCenter::DEFAULT_PORT = 7899 [static]

Default port used for messages.

const int BiometricEvaluation::Process::MessageCenter::DEFAULT_TIMEOUT = 1 [static]

Default number of seconds to wait between polls.

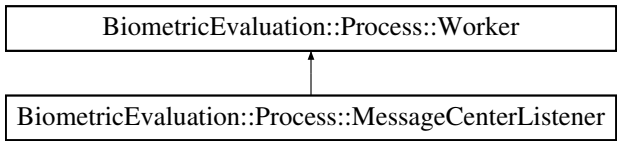
const uint64_t BiometricEvaluation::Process::MessageCenter::MAX_MESSAGE_LENGTH = 255 [static]

Maximum length of a message.

H.67 BiometricEvaluation::Process::MessageCenterListener Class Reference

`#include <be_process_mclistener.h>`

Inheritance diagram for BiometricEvaluation::Process::MessageCenterListener:



Public Member Functions

- int32_t workerMain ()

  The method that will get called to start execution by a ProcessManager.

Static Public Attributes

- static const std::string PARAM_PORT

Additional Inherited Members

H.67.1 Detailed Description

Accepts new connections and spawns message receivers.

H.67.2 Member Function Documentation

int32_t BiometricEvaluation::Process::MessageCenterListener::workerMain ( ) [virtual]

The method that will get called to start execution by a ProcessManager.

Returns

Status code.

Note

If an object of this class is added to a Process::ForkManager object, the implementation of Process::↩ Worker::workerMain() should release all resources prior to returning.

Implements BiometricEvaluation::Process::Worker.

H.67.3 Member Data Documentation

const std::string BiometricEvaluation::Process::MessageCenterListener::PARAM_PORT [static]

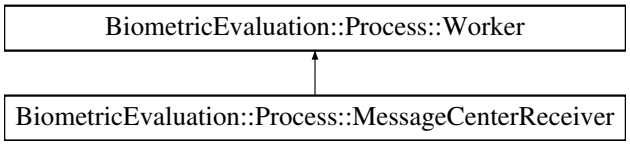
Parameter used to pass port number

H.68 BiometricEvaluation::Process::MessageCenterReceiver Class Reference

Receives message from a client, forwarding to the central MessageCenter.

`#include <be_process_mcreceiver.h>`

Inheritance diagram for BiometricEvaluation::Process::MessageCenterReceiver:



Public Member Functions

- int32_t workerMain ()
- MessageCenterReceiver ()=default
- ~MessageCenterReceiver ()=default

Static Public Attributes

- static const std::string PARAM_CLIENT_SOCKET
- static const std::string PARAM_CLIENT_ID
- static const std::string MSG_DISCONNECT

Additional Inherited Members

H.68.1 Detailed Description

Receives message from a client, forwarding to the central MessageCenter.

H.68.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::MessageCenterReceiver::MessageCenterReceiver ( ) [default]

Default constructor.

BiometricEvaluation::Process::MessageCenterReceiver::~MessageCenterReceiver ( ) [default]

Default destructor.

H.68.3 Member Function Documentation

int32_t BiometricEvaluation::Process::MessageCenterReceiver::workerMain ( ) [virtual]

Receive loop.

Implements BiometricEvaluation::Process::Worker.

H.68.4 Member Data Documentation

const std::string BiometricEvaluation::Process::MessageCenterReceiver::MSG_DISCONNECT [static]

Message sent when client should disconnect.

const std::string BiometricEvaluation::Process::MessageCenterReceiver::PARAM_CLIENT_ID [static]

Parameter used to pass an ID to the client.

const std::string BiometricEvaluation::Process::MessageCenterReceiver::PARAM_CLIENT_SOCKET [static]

Parameter used to pass client socket FD.

H.69 BiometricEvaluation::MPI::MessageTag Class Reference

The types of messages sent between MPI task processes.

`#include <be_mpi.h>`

Public Types

- enum Kind { **Control** = 0, Data = 1, OOB = 2 }

H.69.1 Detailed Description

The types of messages sent between MPI task processes.

H.69.2 Member Enumeration Documentation

enum BiometricEvaluation::MPI::MessageTag::Kind

Enumerator

Data A control message (start, exit, etc.

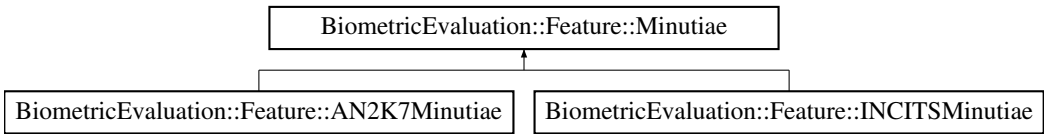
OOB A data message.

H.70 BiometricEvaluation::Feature::Minutiae Class Reference

A class to represent a set of minutiae data points.

`#include <be_feature_minutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::Minutiae:



Public Member Functions

- virtual MinutiaeFormat getFormat () const =0

  Obtain the minutiae format kind.
- virtual MinutiaPointSet getMinutiaPoints () const =0

  Obtain the set of finger minutiae data points. The set may be empty.
- virtual RidgeCountItemSet getRidgeCountItems () const =0

  Obtain the set of ridge count data items. The set may be empty.
- virtual CorePointSet getCores () const =0

  Obtains the set of core positions. The set may be empty.
- virtual DeltaPointSet getDeltas () const =0

  Obtains the set of delta positions. The set may be empty.

H.70.1 Detailed Description

A class to represent a set of minutiae data points.

Each set includes the core and delta data points, if they are included in the source record. This class represents an interface that subclasses of this class will implement, providing more information on the minutioe that is specific to the record format represented by that class.

H.71 BiometricEvaluation::Feature::MinutiaPoint Struct Reference

Representation of a finger minutiae data point.

`#include <be_feature_minutiae.h>`

Public Attributes

- unsigned int **index**
- bool **has_type**
- MinutiaeType **type**
- Image::Coordinate **coordinate**
- unsigned int **theta**
- bool **has_quality**
- unsigned int **quality**

H.71.1 Detailed Description

Representation of a finger minutiae data point.

H.72 BiometricEvaluation::Feature::MPEGFacePoint Struct Reference

Representation of a feature point and a set of points.

`#include <be_feature_mpegfacepoint.h>`

Public Attributes

- uint8_t **type**
- uint8_t **major**
- uint8_t **minor**
- BiometricEvaluation::Image::Coordinate **coordinate**

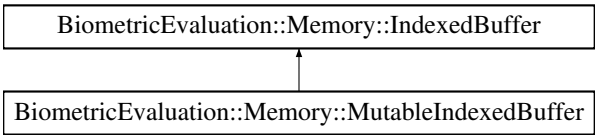
H.72.1 Detailed Description

Representation of a feature point and a set of points.

H.73 BiometricEvaluation::Memory::MutableIndexedBuffer Class Reference

`#include <be_memory_mutableindexedbuffer.h>`

Inheritance diagram for BiometricEvaluation::Memory::MutableIndexedBuffer:



Public Member Functions

- MutableIndexedBuffer (uint8_t *data, uint64_t size)

  Wrap an existing buffer of a given length.
- MutableIndexedBuffer (uint8Array &aa)

  Wrap an existing uint8Array.
- MutableIndexedBuffer (const MutableIndexedBuffer &copy)=default
- uint64_t push (const void *buf, uint64_t len)

  Push elements into the buffer, inreasing the index.
- uint8_t pushU8Val (uint8_t val)

  Push an element into the managed buffer at the current index, incrementing the index.
- uint16_t pushU16Val (uint16_t val)

  Push two elements into the managed buffer at the current index, incrementing the index.
- uint16_t pushBeU16Val (uint16_t val)

  Push two elements into the managed buffer at the current index as a big endian value, incrementing the index.

- uint32_t pushU32Val (uint32_t val)

  Push four elements into the managed buffer at the current index, incrementing the index.
- uint32_t pushBeU32Val (uint32_t val)

  Push four elements into the managed buffer at the current index as a big endian value, incrementing the index.
- uint64_t pushU64Val (uint64_t val)

  Push eight elements into the managed buffer at the current index, incrementing the index.
- virtual const uint8_t * get () const

  Returns a pointer to the managed buffer.
- virtual ~MutableIndexedBuffer ()=default

H.73.1 Detailed Description

Mutable version of an IndexedBuffer.

H.73.2 Constructor & Destructor Documentation

BiometricEvaluation::Memory::MutableIndexedBuffer::MutableIndexedBuffer (uint8_t * *data*, uint64_t *size*)

Wrap an existing buffer of a given length.
Parameters

data	Buffer to wrap.
size	Size of buffer.

BiometricEvaluation::Memory::MutableIndexedBuffer::MutableIndexedBuffer (uint8Array & *aa*)

Wrap an existing uint8Array.
Parameters

aa	uint8Array to wrap.

BiometricEvaluation::Memory::MutableIndexedBuffer::MutableIndexedBuffer (const MutableIndexedBuffer & *copy*) `[default]`

Copy constructor (default).

virtual BiometricEvaluation::Memory::MutableIndexedBuffer::~MutableIndexedBuffer ( ) `[virtual]`, `[default]`

Destructor (default).

H.73.3 Member Function Documentation

virtual const uint8_t* BiometricEvaluation::Memory::MutableIndexedBuffer::get ( ) const `[virtual]`

Returns a pointer to the managed buffer.

Returns

Pointer to the managed buffer.

Reimplemented from BiometricEvaluation::Memory::IndexedBuffer.

uint64_t BiometricEvaluation::Memory::MutableIndexedBuffer::push (const void * *buf*, uint64_t *len*)

Push elements into the buffer, inreasing the index.

Parameters

in	*buf*	The buffer to push. If nullptr, 0 will be inserted.
in	*len*	The number of elements from buf to copy.

Exceptions

Error::DataError	Not enough room to copy len elements.

Returns

The number of elements copied.

uint16_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushBeU16Val (uint16_t *val*)

Push two elements into the managed buffer at the current index as a big endian value, incrementing the index.
Parameters

val	Value to push.

Exceptions

Error::DataError	Not enough room to copy the elements.

Returns

The number of elements copied (2).

uint32_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushBeU32Val (uint32_t *val*)

Push four elements into the managed buffer at the current index as a big endian value, incrementing the index.
Parameters

val	Value to push.

Exceptions

Error::DataError	Not enough room to copy the elements.

Returns

The number of elements copied (4).

uint16_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushU16Val (uint16_t *val*)

Push two elements into the managed buffer at the current index, incrementing the index.
Parameters

val	Value to push.

Exceptions

Error::DataError	Not enough room to copy the elements.

Returns

The number of elements copied (2).

uint32_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushU32Val (uint32_t *val*)

Push four elements into the managed buffer at the current index, incrementing the index.

Parameters

val	Value to push.

Exceptions

Error::DataError	Not enough room to copy the elements.

Returns

The number of elements copied (4).

uint64_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushU64Val (uint64_t *val*)

Push eight elements into the managed buffer at the current index, incrementing the index.

Parameters

val	Value to push.

Exceptions

Error::DataError	Not enough room to copy the elements.

Returns

The number of elements copied (8).

uint8_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushU8Val (uint8_t *val*)

Push an element into the managed buffer at the current index, incrementing the index.

Parameters

val	Value to push.

Exceptions

Error::DataError	Not enough room to copy the element.

Returns

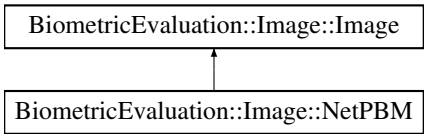
The number of elements copied (1).

H.74 BiometricEvaluation::Image::NetPBM Class Reference

A NetPBM-encoded image.

`#include <be_image_netpbm.h>`

Inheritance diagram for BiometricEvaluation::Image::NetPBM:



Public Types

- enum **Kind** {
  ASCIIPortableBitmap = 1, **ASCIIPortableGraymap** = 2, **ASCIIPortablePixmap** = 3, **BinaryPortableBitmap** = 4,
  BinaryPortableGraymap = 5, **BinaryPortablePixmap** = 6 }

Public Member Functions

- **NetPBM** (const uint8_t *data, const uint64_t size)
- Memory::uint8Array getRawData () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- Memory::uint8Array getRawGrayscaleData (uint8_t depth=8) const

  Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isNetPBM (const uint8_t *data, uint64_t size)
- static void skipLine (const uint8_t *data, size_t dataSize, size_t &offset)

  Skip an entire line of input, placing offset at the first character after the newline.
- static void skipComment (const uint8_t *data, size_t dataSize, size_t &offset)

  Skip a block of comments in input.
- static std::string getNextValue (const uint8_t *data, size_t dataSize, size_t &offset, size_t sizeOfValue=0)

  Obtain the next space-separated value from data, beginning at offset.
- static Memory::uint8Array ASCIIBitmapTo8Bit (const uint8_t *bitmap, uint64_t bitmapSize, uint32_t width, uint32_t height)

  Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.
- static Memory::uint8Array ASCIIPixmapToBinaryPixmap (const uint8_t *ASCIIBuf, uint64_t ASCIIBufSize, uint32_t width, uint32_t height, uint8_t depth, uint32_t maxColor)

  Convert an ASCII pixel map buffer into a binary pixel map buffer.
- static Memory::uint8Array BinaryBitmapTo8Bit (const uint8_t *bitmap, uint64_t bitmapSize, uint32_t width, uint32_t height)

  Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Additional Inherited Members

H.74.1 Detailed Description

A NetPBM-encoded image.

Note

While a NetPBM file can contain more than one image, this class will only support the first image found in any file, also known as the ”plain” NetPBM format.

H.74.2 Member Function Documentation

static Memory::uint8Array BiometricEvaluation::Image::NetPBM::ASCIIBitmapTo8Bit (const uint8_t * *bitmap*, uint64_t *bitmapSize*, uint32_t *width*, uint32_t *height*) [static]

Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.
Parameters

bitmap	Bitmap data buffer.
bitmapSize	Size of bitmap.
width	Width of image in bitmap.
height	Height of image in bitmap.

Returns

8-bit depth representation of bitmap

Exceptions

out_of_range	Error extracting a value from the bitmap.

static Memory::uint8Array BiometricEvaluation::Image::NetPBM::ASCIIPixmapToBinaryPixmap (const uint8_t * *ASCIIBuf*, uint64_t *ASCIIBufSize*, uint32_t *width*, uint32_t *height*, uint8_t *depth*, uint32_t *maxColor*) [static]

Convert an ASCII pixel map buffer into a binary pixel map buffer.
Parameters

ASCIIBuf	ASCII pixel map data buffer.
ASCIIBufSize	Size of ASCIIBuf.
width	Width of image in pixel map.
height	Height of image in pixel map.
depth	Depth of image in pixel map.
maxColor	Maximum color value per pixel. Intensities will be scaled based on this value.

Returns

Binary pixel map representation of the ASCII pixel map in the same depth as the original.

Exceptions

out_of_range	Error extracting a value from the pixel map.
Error::ParameterError	Invalid value for depth, must be a multiple of Image::bitsPerComponent.

static Memory::uint8Array BiometricEvaluation::Image::NetPBM::BinaryBitmapTo8Bit (const uint8_t * *bitmap*, uint64_t *bitmapSize*, uint32_t *width*, uint32_t *height*) [static]

Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Parameters

bitmap	Bitmap data buffer.
bitmapSize	Size of bitmap.
width	Width of image in bitmap.
height	Height of image in bitmap.

Returns

8-bit depth representation of bitmap

Exceptions

out_of_range	Error extracting a value from the bitmap.

static std::string BiometricEvaluation::Image::NetPBM::getNextValue (const uint8_t * *data*, size_t *dataSize*, size_t & *offset*, size_t *sizeOfValue* = *0*) [static]

Obtain the next space-separated value from data, beginning at offset.

Parameters

data	Buffer where next value will be obtained.
dataSize	Size of data.
offset	Current starting position within data.
sizeOfValue	In the event that the values in data are not space-separated, return a value when it reaches sizeOfValue length. 0 assumes space-separated.

Returns

Next value from data.

Memory::uint8Array BiometricEvaluation::Image::NetPBM::getRawData ( ) const [virtual]

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError	Error decompressing image data.
Error::NotImplemented	Compression type not supported.

Note

The raw data returned from this method is encoded at the same bit depth as the compressed data, except in the case of 1-bit (bitmap) images, which are expanded to 8-bit.

Implements BiometricEvaluation::Image::Image.

Memory::uint8Array BiometricEvaluation::Image::NetPBM::getRawGrayscaleData (uint8_t *depth* = *8*) const [virtual]

Accessor for decompressed data in grayscale.
Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 8 or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError	Error decompressing image data.
Error::ParameterError	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

static bool BiometricEvaluation::Image::NetPBM::isNetPBM (const uint8_t * *data*, uint64_t *size*) [static]

Whether or not data is a netpbm image.
Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a netpbm image, false otherwise.

static void BiometricEvaluation::Image::NetPBM::skipComment (const uint8_t * *data*, size_t *dataSize*, size_t & *offset*) [static]

Skip a block of comments in input.

Parameters

data	Buffer with comment to be skipped.
dataSize	Size of data
offset	Position within data from which the rest of the line should be read.

Exceptions

out_of_range	End of line not encountered before end of data or on last line of data.

static void BiometricEvaluation::Image::NetPBM::skipLine (const uint8_t * *data*, size_t *dataSize*, size_t & *offset*) [static]

Skip an entire line of input, placing offset at the first character after the newline.

Parameters

data	Buffer with line to be skipped.
dataSize	Size of data.
offset	Position within data from which the rest of the line should be read.

Exceptions

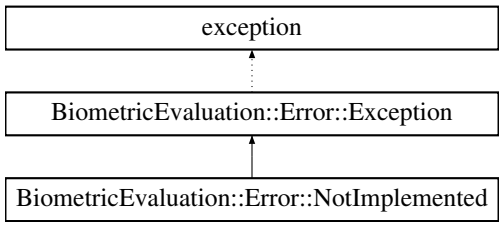
out_of_range	End of line not encountered before end of data or on last line of data.

H.75 BiometricEvaluation::Error::NotImplemented Class Reference

A NotImplemented object is thrown when the underlying implementation of this interface has not or could not be created.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::NotImplemented:



Public Member Functions

- NotImplemented ()
- NotImplemented (std::string info)

H.75.1 Detailed Description

A NotImplemented object is thrown when the underlying implementation of this interface has not or could not be created.

H.75.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::NotImplemented::NotImplemented ( )

Construct a NotImplemented object with the default information string.

BiometricEvaluation::Error::NotImplemented::NotImplemented (std::string *info*)

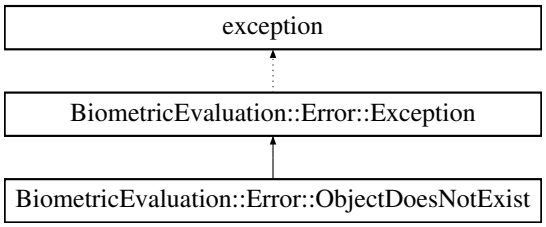
Construct a NotImplemented object with an information string appended to the default information string.

H.76 BiometricEvaluation::Error::ObjectDoesNotExist Class Reference

The named object does not exist.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectDoesNotExist:



Public Member Functions

- ObjectDoesNotExist ()
- ObjectDoesNotExist (std::string info)

H.76.1 Detailed Description

The named object does not exist.

H.76.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( )

Construct a ObjectDoesNotExist object with the default information string.

BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist (std::string *info*)

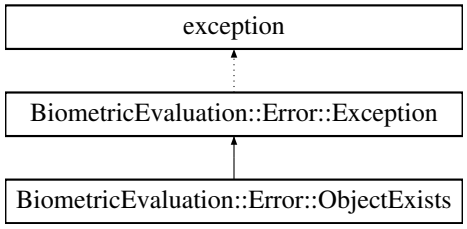
Construct a ObjectDoesNotExist object with an information string appended to the default information string.

H.77 BiometricEvaluation::Error::ObjectExists Class Reference

The named object exists and will not be replaced.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectExists:



Public Member Functions

- ObjectExists ()
- ObjectExists (std::string info)

H.77.1 Detailed Description

The named object exists and will not be replaced.

H.77.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ObjectExists::ObjectExists ( )

Construct a ObjectExists object with the default information string.

BiometricEvaluation::Error::ObjectExists::ObjectExists (std::string *info*)

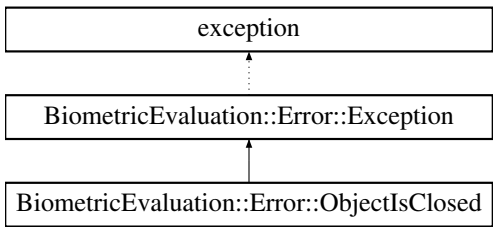
Construct a ObjectExists object with an information string appended to the default information string.

H.78 BiometricEvaluation::Error::ObjectIsClosed Class Reference

The object is closed.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectIsClosed:



Public Member Functions

- ObjectIsClosed ()
- ObjectIsClosed (std::string info)

H.78.1 Detailed Description

The object is closed.

H.78.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ( )

Construct a ObjectIsClosed object with the default information string.

BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed (std::string *info*)

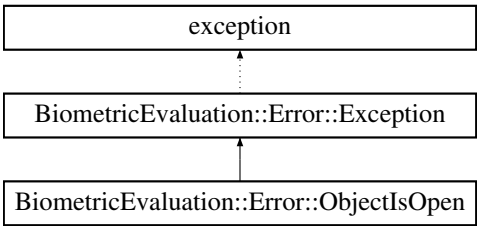
Construct a ObjectIsClosed object with an information string appended to the default information string.

H.79 BiometricEvaluation::Error::ObjectIsOpen Class Reference

The object is already opened.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectIsOpen:



Public Member Functions

- ObjectIsOpen ()
- ObjectIsOpen (std::string info)

H.79.1 Detailed Description

The object is already opened.

H.79.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ( )

Construct a ObjectIsOpen object with the default information string.

BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen (std::string *info*)

Construct a ObjectIsOpen object with an information string appended to the default information string.

H.80 BiometricEvaluation::Memory::OrderedMap< Key, T > Class Template Reference

`#include <be_memory_orderedmap.h>`

Public Types

- using **container** = typename std::unordered_map< Key, T >
- using **iterator** = OrderedMapIterator< Key, T >
- using **const_iterator** = OrderedMapConstIterator< Key, T >
- using **size_type** = typename container::size_type
- using **value_type** = typename container::value_type
- using **key_type** = Key
- using **mapped_type** = T
- using **key_equal** = typename container::key_equal

Public Member Functions

- OrderedMap ()
- bool push_back (const value_type &value)

  Insert an element at the end of the collection.
- void erase (iterator pos)

  Remove an element from the collection.
- void erase (const Key &key)

  Remove an element from the collection.
- iterator begin ()
- const_iterator begin () const
- const_iterator cbegin () const
- iterator end ()
- const_iterator end () const
- const_iterator cend () const
- size_type size () const
- bool keyExists (const Key &key) const

  Determine if a value exists in the container.
- const OrderedMapIterator< Key, T > find (const Key &key) const

  Obtain an iterator to a particular key.
- std::shared_ptr< value_type > **find_quick** (const Key &key) const
- T & operator[ ] (const Key &key)

  Subscripting operator.
- key_equal key_eq () const
- ~OrderedMap ()

Friends

- class **OrderedMapIterator< Key, T >**
- class **OrderedMapConstIterator< Key, T >**

H.80.1 Detailed Description

template<class Key, class T>class BiometricEvaluation::Memory::OrderedMap< Key, T >

A map where insertion order is preserved and elements are unique.

H.80.2 Constructor & Destructor Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::OrderedMap ( )

Constructor.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::~OrderedMap ( )

Destructor

H.80.3 Member Function Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::begin ( )

Returns

Iterator at the first element of the collection.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::begin ( ) const

Returns

Iterator at the first element of the collection.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::cbegin ( ) const

Returns

Iterator at the first element of the collection.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::cend ( ) const

Returns

Iterator beyond the last element of the collection.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::end ( )

Returns

Iterator beyond the last element of the collection.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::end ( ) const

Returns

Iterator beyond the last element of the collection.

template<class Key , class T > void BiometricEvaluation::Memory::OrderedMap< Key, T >::erase (iterator *pos*)

Remove an element from the collection.

Parameters

pos	Iterator to element at the position which should be removed.

Note

Complexity: Average case: O(1), worst case O(size()).

template<class Key, class T > void BiometricEvaluation::Memory::OrderedMap< Key, T >::erase (const Key & *key*)

Remove an element from the collection.
Parameters

key	Key of the element to remove.

template<class Key, class T > const BiometricEvaluation::Memory::OrderedMapIterator< Key, T > BiometricEvaluation::Memory::OrderedMap< Key, T >::find (const Key & *key*) const

Obtain an iterator to a particular key.

Note

Complexity is O(n).

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::key_equal BiometricEvaluation::Memory::OrderedMap< Key, T >::key_eq ( ) const

Returns

Function that compares keys for equality.

template<class Key, class T > bool BiometricEvaluation::Memory::OrderedMap< Key, T >::keyExists (const Key & *key*) const

Determine if a value exists in the container.
Parameters

key	Key to search the container for.

Returns

Whether or not key exists in this container.

Note

Complexity is O(1).

template<class Key, class T > T & BiometricEvaluation::Memory::OrderedMap< Key, T >::operator[ ] (const Key & *key*)

Subscripting operator.

Parameters

key	Key used to index into the map.

Returns

Value for key, which may be a new value.

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMap< Key, T >::push_back (const value_type & *value*)

Insert an element at the end of the collection.
Parameters

value	Value to insert.

Returns

Whether or not the object was inserted.

Note

Complexity: Average case: O(1), worst case O(size()).

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::size_type BiometricEvaluation::Memory::OrderedMap< Key, T >::size ( ) const

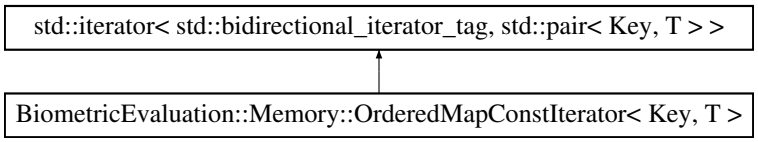
Returns

Number of elements in the collection.

H.81 BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > Class Template Reference

`#include <be_memory_orderedmap.h>`

Inheritance diagram for BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >:



Public Types

- using **reference** = typename std::iterator_traits< OrderedMapConstIterator >::reference
- using **const_reference** = const typename std::iterator_traits< OrderedMapConstIterator >::reference
- using **pointer** = typename std::iterator_traits< OrderedMapConstIterator >::pointer
- using **const_pointer** = const typename std::iterator_traits< OrderedMapConstIterator >::pointer
- using **value_type** = typename std::iterator_traits< OrderedMapConstIterator >::value_type
- using **difference_type** = typename std::iterator_traits< OrderedMapConstIterator >::difference_type

Public Member Functions

- OrderedMapConstIterator ()
- OrderedMapConstIterator (const OrderedMapIterator< Key, T > &iterator)
- ~OrderedMapConstIterator ()
- const_reference operator* () const
- const_pointer operator-> () const
- OrderedMapConstIterator & operator++ ()
- OrderedMapConstIterator & operator++ (int dummy)
- OrderedMapConstIterator & operator-- ()
- OrderedMapConstIterator & operator-- (int dummy)
- bool operator== (const OrderedMapConstIterator &rhs) const

  Test for iterator equality.
- bool operator!= (const OrderedMapConstIterator &rhs) const

  Test for iterator equality.

Friends

- class **OrderedMap< Key, T >**

H.81.1 Detailed Description

template<class Key, class T>class BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >

Const Iterator for OrderedMaps.

H.81.2 Constructor & Destructor Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::OrderedMapConstIterator ( )

Constructor

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::OrderedMapConstIterator (const OrderedMapIterator< Key, T > & *iterator*)

Iterator to ConstIterator converter

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::~OrderedMapConstIterator ( )

Destructor

H.81.3 Member Function Documentation

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator!= (const OrderedMapConstIterator< Key, T > & *rhs*) const

Test for iterator equality.

Parameters

rhs	Object on the right-hand side of the expression.

Returns

Whether or not this iterator is not equivalent to rhs.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::const_reference BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator* ( ) const

Returns

Reference to the current iterated pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator++ ( )

Move to the next pair

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator++ (int *dummy*)

Move to the next pair

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator-- ( )

Move to the previous pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator-- (int *dummy*)

Move to the previous pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::const_pointer BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator-> ( ) const

Returns

Pointer to the current iterated pair.

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator== (const OrderedMapConstIterator< Key, T > & *rhs*) const

Test for iterator equality.

Parameters

rhs	Object on the right-hand side of the expression.

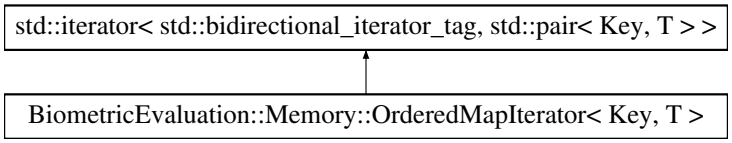
Returns

Whether or not this iterator is equivalent to rhs.

H.82 BiometricEvaluation::Memory::OrderedMapIterator< Key, T > Class Template Reference

`#include <be_memory_orderedmap.h>`

Inheritance diagram for BiometricEvaluation::Memory::OrderedMapIterator< Key, T >:



Public Types

- using **reference** = typename std::iterator_traits< OrderedMapIterator >::reference
- using **pointer** = typename std::iterator_traits< OrderedMapIterator >::pointer
- using **value_type** = typename std::iterator_traits< OrderedMapIterator >::value_type
- using **difference_type** = typename std::iterator_traits< OrderedMapIterator >::difference_type

Public Member Functions

- OrderedMapIterator ()
- ~OrderedMapIterator ()
- reference operator* () const
- pointer operator-> () const
- OrderedMapIterator & operator++ ()
- OrderedMapIterator & operator++ (int dummy)
- OrderedMapIterator & operator-- ()
- OrderedMapIterator & operator-- (int dummy)
- bool operator== (const OrderedMapIterator &rhs) const

  Test for iterator equality.
- bool operator!= (const OrderedMapIterator &rhs) const

  Test for iterator equality.

Friends

- class **OrderedMap< Key, T >**
- class **OrderedMapConstIterator< Key, T >**

H.82.1 Detailed Description

template<class Key, class T>class BiometricEvaluation::Memory::OrderedMapIterator< Key, T >

Iterator for OrderedMaps.

H.82.2 Constructor & Destructor Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::OrderedMapIterator ( )

Constructor

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::~OrderedMapIterator ( )

Destructor

H.82.3 Member Function Documentation

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator!= (const OrderedMapIterator< Key, T > & *rhs*) const

Test for iterator equality.
Parameters

rhs	Object on the right-hand side of the expression.

Returns

Whether or not this iterator is not equivalent to rhs.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::reference BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator* ( ) const

Returns

Reference to the current iterated pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator++ ( )

Move to the next pair

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator++ (int *dummy*)

Move to the next pair

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator-- ( )

Move to the previous pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator-- (int *dummy*)

Move to the previous pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::pointer BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator-> ( ) const

Returns

Pointer to the current iterated pair.

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator== (const OrderedMapIterator< Key, T > & *rhs*) const

Test for iterator equality.
Parameters

rhs	Object on the right-hand side of the expression.

Returns

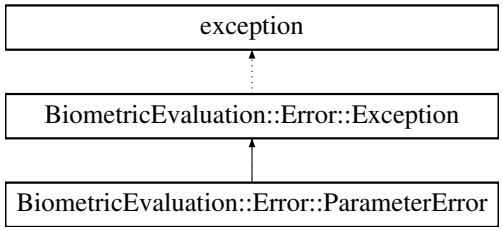
Whether or not this iterator is equivalent to rhs.

H.83 BiometricEvaluation::Error::ParameterError Class Reference

An invalid parameter was passed to a constructor or method.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ParameterError:



Public Member Functions

- ParameterError ()
- ParameterError (std::string info)

H.83.1 Detailed Description

An invalid parameter was passed to a constructor or method.

H.83.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ParameterError::ParameterError ( )

Construct a ParameterError object with the default information string.

BiometricEvaluation::Error::ParameterError::ParameterError (std::string *info*)

Construct a ParameterError object with an information string appended to the default information string.

H.84 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification Class Reference

Pattern classification codes.

`#include <be_feature_an2k7minutiae.h>`

Classes

- struct Entry

Public Types

- using **Entry** = struct Entry

H.84.1 Detailed Description

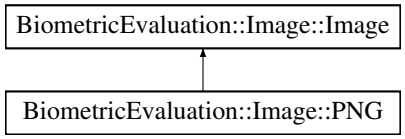
Pattern classification codes.

H.85 BiometricEvaluation::Image::PNG Class Reference

A PNG-encoded image.

`#include <be_image_png.h>`

Inheritance diagram for BiometricEvaluation::Image::PNG:



Public Member Functions

- **PNG** (const uint8_t *data, const uint64_t size)
- Memory::uint8Array getRawData () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- Memory::uint8Array getRawGrayscaleData (uint8_t depth=8) const

  Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isPNG (const uint8_t *data, uint64_t size)

Additional Inherited Members

H.85.1 Detailed Description

A PNG-encoded image.

H.85.2 Member Function Documentation

Memory::uint8Array BiometricEvaluation::Image::PNG::getRawData ( ) const [virtual]

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError	Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

Memory::uint8Array BiometricEvaluation::Image::PNG::getRawGrayscaleData (uint8_t *depth* = 8) const [virtual]

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 8 or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError	Error decompressing image data.
Error::ParameterError	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

static bool BiometricEvaluation::Image::PNG::isPNG (const uint8_t * *data*, uint64_t *size*) [static]

Whether or not data is a PNG image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a PNG image, false otherwise

H.86 BiometricEvaluation::Face::PoseAngle Struct Reference

Representation of pose angle and uncertainty.

`#include <be_face.h>`

Public Attributes

- uint8_t **yaw**
- uint8_t **pitch**
- uint8_t **roll**
- uint8_t **yawUncertainty**
- uint8_t **pitchUncertainty**
- uint8_t **rollUncertainty**

H.86.1 Detailed Description

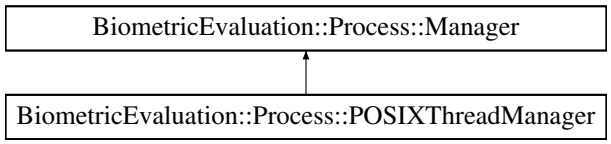
Representation of pose angle and uncertainty.

H.87 BiometricEvaluation::Process::POSIXThreadManager Class Reference

Manager implementation that starts Workers in POSIX threads.

`#include <be_process_posixthreadmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::POSIXThreadManager:



Public Member Functions

- POSIXThreadManager ()
- std::shared_ptr< WorkerController > addWorker (std::shared_ptr< Worker > worker)

  Adds a Worker to be managed by this Manager.
- void startWorkers (bool wait=true, bool communicate=false)

  Begin Worker's work.
- void startWorker (std::shared_ptr< WorkerController > worker, bool wait=true, bool communicate=false)

  Start a Worker.

- int32_t stopWorker (std::shared_ptr< WorkerController > workerController)

  Ask Worker to exit.
- void waitForWorkerExit ()

  Block until all Workers have exited.
- ~POSIXThreadManager ()

  ~POSIXThreadManager destructor.

Additional Inherited Members

H.87.1 Detailed Description

Manager implementation that starts Workers in POSIX threads.

H.87.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::POSIXThreadManager::POSIXThreadManager ( )

POSIXThreadManager constructor.

H.87.3 Member Function Documentation

std::shared_ptr<WorkerController> BiometricEvaluation::Process::POSIXThreadManager::addWorker (std::shared_ptr< Worker > *worker*) [virtual]

Adds a Worker to be managed by this Manager.

Parameters

worker	A Worker instance to run.

Returns

shared_ptr to worker.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::POSIXThreadManager::startWorker (std::shared_ptr< WorkerController > *worker*, bool *wait* = `true`, bool *communicate* = `false`) [virtual]

Start a Worker.

Parameters

worker	Pointer to a WorkerController that is being managed by this Manager instance.
wait	Whether or not to wait for this Worker to exit before returning control to the caller.
communicate	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::POSIXThreadManager::startWorkers (bool *wait* = `true`, bool *communicate* = `false`) [virtual]

Begin Worker's work.

Parameters

in	*wait*	Whether or not to wait for all Workers to return before returning.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	At least one Worker is already working.
Error::StrategyError	Problem starting the Workers.

Implements BiometricEvaluation::Process::Manager.

int32_t BiometricEvaluation::Process::POSIXThreadManager::stopWorker (std::shared_ptr< WorkerController > *workerController*) [virtual]

Ask Worker to exit.
Parameters

workerController	Pointer to the WorkerController that should be stopped.

Returns

Exit status of worker.

Exceptions

Error::ObjectDoesNotExist	worker is not working.
Error::StrategyError	Problem sending the signal.

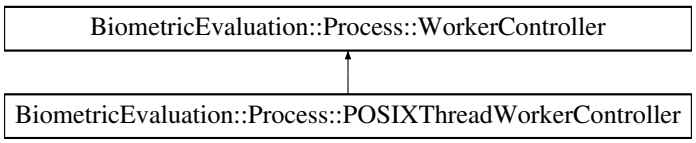
Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::POSIXThreadManager::waitForWorkerExit ( ) [virtual]

Block until all Workers have exited.
Use this method if wait=false was set during a call to startWorker(s) but now wait=true is desired.
Implements BiometricEvaluation::Process::Manager.

H.88 BiometricEvaluation::Process::POSIXThreadWorkerController Class Reference

Decorated Worker returned from a Process::POSIXThreadManager.
`#include <be_process_posixthreadmanager.h>`
Inheritance diagram for BiometricEvaluation::Process::POSIXThreadWorkerController:



Public Member Functions

- void reset ()

  Reuse the Worker.
- bool isWorking () const

  Obtain whether or not Worker is working.
- bool everWorked () const

  Obtain whether or not this Worker has ever worked.
- ~POSIXThreadWorkerController ()

  POSIXThreadWorkerController destructor.

Friends

- class **POSIXThreadManager**

Additional Inherited Members

H.88.1 Detailed Description

Decorated Worker returned from a Process::POSIXThreadManager.

H.88.2 Member Function Documentation

bool BiometricEvaluation::Process::POSIXThreadWorkerController::everWorked ( ) const [virtual]

Obtain whether or not this Worker has ever worked.

Returns

true the Worker has ever or is currently working, false otherwise.

Note

reset() will change the result of this method.

Implements BiometricEvaluation::Process::WorkerController.

bool BiometricEvaluation::Process::POSIXThreadWorkerController::isWorking ( ) const [virtual]

Obtain whether or not Worker is working.

Returns

Whether or not the Worker is working.

Implements BiometricEvaluation::Process::WorkerController.

void BiometricEvaluation::Process::POSIXThreadWorkerController::reset ( ) [virtual]

Reuse the Worker.

Exceptions

Error::ObjectExists	The previously started Worker is still running.

Reimplemented from BiometricEvaluation::Process::WorkerController.

H.89 BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate Struct Reference

Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

`#include <be_finger_an2kview_varres.h>`

Public Member Functions

- PrintPositionCoordinate (FingerImageCode &fingerView, FingerImageCode &segment, Image::CoordinateSet &coordinates)

  Construct a PrintPositionCoordinate.

Public Attributes

- FingerImageCode fingerView
- FingerImageCode segment
- Image::CoordinateSet coordinates

H.89.1 Detailed Description

Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

H.89.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::PrintPositionCoordinate (FingerImageCode & *fingerView,* FingerImageCode & *segment,* Image::CoordinateSet & *coordinates*)

Construct a PrintPositionCoordinate.

Parameters

fingerView	The full finger view being referred to.
segment	Location of a segment within fingerView. If segment is NA, the image referred to is the entire image or tip.
coordinates	Two coordinates creating a bounding rectangle (top left vertex, lower right vertex).

H.89.3 Member Data Documentation

Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::coordinates

Two coordinates forming bounding box

FingerImageCode BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::fingerView

Full finger view being bounded

FingerImageCode BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPosition↩ Coordinate::segment

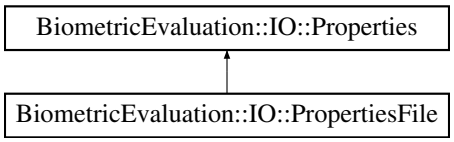
Segment within full finger view bound

H.90 BiometricEvaluation::IO::Properties Class Reference

Maintain key/value pairs of strings, with each property matched to one value.

`#include <be_io_properties.h>`

Inheritance diagram for BiometricEvaluation::IO::Properties:



Public Member Functions

- Properties (uint8_t mode=IO::READWRITE)

  Construct a new Properties object.
- Properties (const uint8_t *buffer, const size_t size, uint8_t mode=IO::READWRITE)

  Construct a new Properties object from the contents of a buffer.
- virtual void setProperty (const std::string &property, const std::string &value)

  Set a property with a value.
- virtual void setPropertyFromInteger (const std::string &property, int64_t value)

  Set a property with an integer value.
- virtual void setPropertyFromDouble (const std::string &property, double value)

  Set a property with a double value.
- virtual void setPropertyFromBoolean (const std::string &property, bool value)

  Set a property with a boolean value.
- virtual void removeProperty (const std::string &property)

  Remove a property.
- virtual std::string getProperty (const std::string &property) const

  Retrieve a property value as a string object.
- virtual int64_t getPropertyAsInteger (const std::string &property) const

  Retrieve a property value as an integer value.
- virtual double getPropertyAsDouble (const std::string &property) const

  Retrieve a property value as a double value.
- virtual bool **getPropertyAsBoolean** (const std::string &property) const
- std::vector< std::string > getPropertyKeys () const

  Retrieve a set of all property keys.
- virtual ~Properties ()

Protected Member Functions

- uint8_t getMode () const

  Obtain the mode of the Properties object.
- void initWithBuffer (const Memory::uint8Array &buffer)

  Initialize the PropertiesMap with the contents of a properly formatted buffer.
- void initWithBuffer (const uint8_t *const buffer, size_t size)

  Initialize the PropertiesMap with the contents of a properly formatted buffer.

H.90.1 Detailed Description

Maintain key/value pairs of strings, with each property matched to one value.

H.90.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::Properties::Properties (uint8_t *mode* = *IO::READWRITE*)

Construct a new Properties object.

Parameters

in	*mode*	The read/write mode of the object.

BiometricEvaluation::IO::Properties::Properties (const uint8_t * *buffer*, const size_t *size*, uint8_t *mode* = *IO::READWRITE*)

Construct a new Properties object from the contents of a buffer.

The format of the buffer can be seen in PropertiesFile.

Parameters

in	*buffer*	A buffer that contains the contents of a Property file.
in	*size*	The size of buffer.
in	*mode*	The read/write mode of the object.

Exceptions

Error::StrategyError	A line in the properties file is malformed.

virtual BiometricEvaluation::IO::Properties::~Properties ( ) [virtual]

Destructor

H.90.3 Member Function Documentation

uint8_t BiometricEvaluation::IO::Properties::getMode ( ) const [protected]

Obtain the mode of the Properties object.

Returns

Mode (IO::READONLY or IO::READWRITE)

virtual std::string BiometricEvaluation::IO::Properties::getProperty (const std::string & *property*) const [virtual]

Retrieve a property value as a string object.

Parameters

in	*property*	The name of the property to get.

Exceptions

Error::ObjectDoesNot↩ Exist	The named property does not exist.

virtual double BiometricEvaluation::IO::Properties::getPropertyAsDouble (const std::string & *property*) const [virtual]

Retrieve a property value as a double value.

Parameters

in	*property*	The name of the property to get.

Exceptions

Error::ObjectDoesNot↩ Exist	The named property does not exist.

virtual int64_t BiometricEvaluation::IO::Properties::getPropertyAsInteger (const std::string & *property*) const [virtual]

Retrieve a property value as an integer value.

Integer value strings for properties can represent either decimal or hexadecimal values, which must be preceded with either "0x" or "0X".

Parameters

in	*property*	The name of the property to get.

Exceptions

Error::ObjectDoesNot↩ Exist	The named property does not exist.
Error::ConversionError	The property value cannot be converted, usually due to non-numeric characters in the string.

std::vector<std::string> BiometricEvaluation::IO::Properties::getPropertyKeys ( ) const

Retrieve a set of all property keys.

Returns

A vector of property key strings.

void BiometricEvaluation::IO::Properties::initWithBuffer (const Memory::uint8Array & *buffer*) [protected]

Initialize the PropertiesMap with the contents of a properly formatted buffer.

This method ensures that the PropertiesMap contains only the properties found within the buffer.

Parameters

buffer	Contents of a properties file.

Exceptions

Error::StrategyError	A line of the buffer is malformed.

void BiometricEvaluation::IO::Properties::initWithBuffer (const uint8_t *const *buffer*, size_t *size*) [protected]

Initialize the PropertiesMap with the contents of a properly formatted buffer.

This method ensures that the PropertiesMap contains only the properties found within the buffer.

Parameters

buffer	Contents of a properties file.
size	Size of the buffer.

Exceptions

Error::StrategyError	A line of the buffer is malformed.

virtual void BiometricEvaluation::IO::Properties::removeProperty (const std::string & *property*) [virtual]

Remove a property.

Parameters

in	*property*	The name of the property to set.

Exceptions

Error::ObjectDoesNotExist	The named property does not exist.
Error::StrategyError	The Properties object is read-only.

virtual void BiometricEvaluation::IO::Properties::setProperty (const std::string & *property*, const std::string & *value*) [virtual]

Set a property with a value.

Both the property and value will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise, the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError	The Properties object is read-only.

virtual void BiometricEvaluation::IO::Properties::setPropertyFromBoolean (const std::string & *property*, bool *value*) [virtual]

Set a property with a boolean value.

The actual value to be written is implementation- defined and may not actually be preserved, but the boolean value is guaranteed to remain valid when read with getPropertyAsBoolean().

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError	The Properties object is read-only.

virtual void BiometricEvaluation::IO::Properties::setPropertyFromDouble (const std::string & *property*, double *value*) [virtual]

Set a property with a double value.

The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError	The Properties object is read-only.

virtual void BiometricEvaluation::IO::Properties::setPropertyFromInteger (const std::string & *property*, int64_t *value*) [virtual]

Set a property with an integer value.

The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

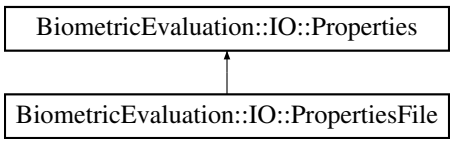
Error::StrategyError	The Properties object is read-only.

H.91 BiometricEvaluation::IO::PropertiesFile Class Reference

A Properties object persisted in an file on disk.

`#include <be_io_propertiesfile.h>`

Inheritance diagram for BiometricEvaluation::IO::PropertiesFile:



Public Member Functions

- PropertiesFile (const std::string &pathname, uint8_t mode=IO::READWRITE)

  Construct a new Properties object from an existing or to be created properties file. The constructor will create the file when it does not exist.
- void sync ()

  Write the properties to the underlying file, synchronizing the in-memory and on-disk versions.
- void changeName (const std::string &pathname)

  Change the name of the Properties, which means changing the name of the underlying file that stores the properties.
- ~PropertiesFile ()
- PropertiesFile (const PropertiesFile &other)=delete

  Copy constructor (disabled).
- PropertiesFile & operator= (const PropertiesFile &other)=delete

  Assignment operator (disabled).

Additional Inherited Members

H.91.1 Detailed Description

A Properties object persisted in an file on disk.
An example file might look like this:

```
*     Name = John Smith
*     Age = 32
*     Favorite Hex Number = 0xffff
*
```

For property keys and values, leading and trailing whitespace is removed, therefore the call

```
props->setProperty("  My property   ", "   A Value  ");
```

results in an entry in the property file as

```
*     My property = A value
*
```

Therefore, the property names "Foo", " Foo", "Foo " are equivalent.

H.91.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::PropertiesFile::PropertiesFile (const std::string & *pathname*, uint8_t *mode* = *IO::READWRITE*)

Construct a new Properties object from an existing or to be created properties file. The constructor will create the file when it does not exist.
Parameters

in	*pathname*	The path to the file to store the properties.
in	*mode*	The read/write mode of the object.

Exceptions

Error::StrategyError	A line in the properties file is malformed.
Error::FileError	An error occurred when using the underlying storage system.

BiometricEvaluation::IO::PropertiesFile::~PropertiesFile ( )

Destructor

BiometricEvaluation::IO::PropertiesFile::PropertiesFile (const PropertiesFile & *other*) `[delete]`

Copy constructor (disabled).

Disabled because this object could represent a file on disk.

Parameters

other	PropertiesFile object to copy.

H.91.3 Member Function Documentation

void BiometricEvaluation::IO::PropertiesFile::changeName (const std::string & *pathname*)

Change the name of the Properties, which means changing the name of the underlying file that stores the properties.

Note

No check is made that the file is writeable at this time.

Parameters

`in`	*pathname*	The path to the Properties file.

Exceptions

Error::StrategyError	The object is read-only.
Error::ObjectExists	A file at pathname already exists.

PropertiesFile& BiometricEvaluation::IO::PropertiesFile::operator= (const PropertiesFile & *other*) `[delete]`

Assignment operator (disabled).

Disabled because this object could represent a file on disk.

Parameters

other	PropertiesFile object to assign;

Returns

This PropertiesFile object, now containing the contents of other.

void BiometricEvaluation::IO::PropertiesFile::sync ( )

Write the properties to the underlying file, synchronizing the in-memory and on-disk versions.

Exceptions

Error::FileError	An error occurred when using the underlying storage system.
Error::StrategyError	The object was constructed with nullptr as the file name, or is read-only.

H.92 BiometricEvaluation::Iris::INCITSView::QualitySubBlock Struct Reference

Representation of an iris quality block.

`#include <be_iris_incitsview.h>`

Public Attributes

- uint8_t **score**
- uint16_t **vendorID**
- uint16_t **algorithmID**

H.92.1 Detailed Description

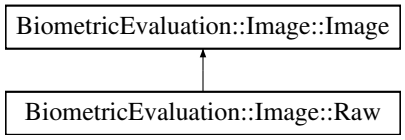
Representation of an iris quality block.

H.93 BiometricEvaluation::Image::Raw Class Reference

An image with no encoding or compression.

`#include <be_image_raw.h>`

Inheritance diagram for BiometricEvaluation::Image::Raw:



Public Member Functions

- **Raw** (const uint8_t *data, const uint64_t size, const Size dimensions, const unsigned int depth, const Resolution resolution)
- Memory::uint8Array getRawData () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- Memory::uint8Array getRawGrayscaleData (uint8_t depth=8) const

  Accessor for decompressed data in grayscale.

Additional Inherited Members

H.93.1 Detailed Description

An image with no encoding or compression.

H.93.2 Member Function Documentation

Memory::uint8Array BiometricEvaluation::Image::Raw::getRawData ( ) const [virtual]

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError	Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

Memory::uint8Array BiometricEvaluation::Image::Raw::getRawGrayscaleData (uint8_t *depth* = *8*) const [virtual]

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 8 or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError	Error decompressing image data.
Error::ParameterError	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

H.94 BiometricEvaluation::MPI::Receiver Class Reference

A class to represent an MPI task that receives WorkPackages containers from the Distributor.

`#include <be_mpi_receiver.h>`

Public Member Functions

- Receiver (const std::string &propertiesFileName, const std::shared_ptr< BiometricEvaluation::MPI::↩ WorkPackageProcessor > &workPackageProcessor)

  Construct a new work package receiver.
- void start ()

  Start the receiving task.

H.94.1 Detailed Description

A class to represent an MPI task that receives WorkPackages containers from the Distributor.

A receiver object depends on a set of properties contained in a file. The properties specify MPI settings, and other items. Subclasses of the class can add new properties.

Each receiver object is responsible for 1..n worker processes that are started when Receiver::start() is called. The receiver will start workers only when the distributor indicates that it has started successfully. Otherwise, the Receiver transitions to the shutdown state.

One of the optional properties is a Uniform Resource Locator (URL) for the Logsheet. If this property does not exist, no logging takes place (although applications can create their own Logsheet). If the URL is present, the framework will log at various points of processing. In the case of a FileLogsheet the framework will create more than one log file, each named after the ID of the MPI task created by the MPI runtime, and the child process created by Receiver.

See also

IO::Properties
IO::Logsheet
MPI::Distributor
Process::Worker

H.94.2 Constructor & Destructor Documentation

BiometricEvaluation::MPI::Receiver::Receiver (const std::string & *propertiesFileName,* const std::shared_ptr< BiometricEvaluation::MPI::WorkPackageProcessor > & *workPackageProcessor*)

Construct a new work package receiver.

Parameters

in	*propertiesFile↩ Name*	The name of the file containing the properties used by the receiver object.
in	*workPackage↩ Processor*	The object that will process the work received by this object.

Exceptions

Error::Exception	An error occurred when constructing this object.

H.94.3 Member Function Documentation

void BiometricEvaluation::MPI::Receiver::start ( )

Start the receiving task.

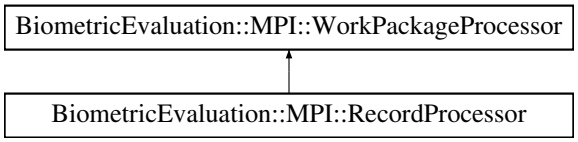
Upon starting, the Receiver object will begin communicating with the Distributor using MPI messages. This Receiver object will send a status message back to the Distributor indicating success or failure to initialize. Success includes the startup of at least one worker process.

H.95 BiometricEvaluation::MPI::RecordProcessor Class Reference

An implementation of a work package processor that will extract record store keys, and optionally, values, from a WorkPackage.

`#include <be_mpi_recordprocessor.h>`

Inheritance diagram for BiometricEvaluation::MPI::RecordProcessor:



Public Member Functions

- RecordProcessor (const std::string &propertiesFileName)

  Construct a work package processor with the given properties.
- virtual void processRecord (const std::string &key)=0

  Method implemented by child classes to perform an action using each record from the Record Store.
- virtual void processRecord (const std::string &key, const Memory::uint8Array &value)=0

  Method implemented by child classes to perform an action using each record from the Record Store.
- virtual std::shared_ptr
  < WorkPackageProcessor > newProcessor (std::shared_ptr< IO::Logsheet > &logsheet)=0

  Obtain an object that will process work packages. This method is part of the factory personality.
- virtual void performInitialization (std::shared_ptr< IO::Logsheet > &logsheet)=0

  Initialization function to be called before work is distributed to the work package processor.
- void processWorkPackage (MPI::WorkPackage &workPackage)

  Process the data contents of the work package. This method is part of the worker personality.

Protected Member Functions

- std::shared_ptr
  < MPI::RecordStoreResources > **getResources** ()

H.95.1 Detailed Description

An implementation of a work package processor that will extract record store keys, and optionally, values, from a WorkPackage.

Subclasses of this abstract class must implement the method to process the records associated with the keys.

H.95.2 Constructor & Destructor Documentation

BiometricEvaluation::MPI::RecordProcessor::RecordProcessor (const std::string & *propertiesFileName*)

Construct a work package processor with the given properties.

A record processor uses a named record store to retrieve the data to be processed when only the key is delivered as part of a work package. When both key and value are part of the work package, there is no need to have access to the source record store.

Note

The size of a single value item is limited to 2^32 octets. If the size of the value item is larger, behavior is undefined.

Parameters

in	*propertiesFile↩ Name*	The name of the file containing the properties for this object.

Exceptions

Error::Exception	An error occurred, usually due to missing or incorrect properties.

H.95.3 Member Function Documentation

virtual std::shared_ptr<WorkPackageProcessor> BiometricEvaluation::MPI::Record↩ Processor::newProcessor (std::shared_ptr< IO::Logsheet > & *logsheet*) [pure virtual]

Obtain an object that will process work packages. This method is part of the factory personality.
Parameters

logsheet	A shared pointer to the IO::Logsheet that may be used to save messages generated by the object.

Returns

A shared pointer to the work package processor.

Implements BiometricEvaluation::MPI::WorkPackageProcessor.

virtual void BiometricEvaluation::MPI::RecordProcessor::performInitialization (std::shared_ptr< IO::Logsheet > & *logsheet*) [pure virtual]

Initialization function to be called before work is distributed to the work package processor.

Implementations of this class can use this function to do any processing necessary before work is given to the processor, pre-forking.

This method is part of the factory personality. All state that is to be common across all package processor objects can be initialized in this method.
Parameters

logsheet	A shared pointer to the IO::Logsheet that may be used to save messages generated by the object.

Exceptions

Error::Exception	An implementation specific. error occurred.

Implements BiometricEvaluation::MPI::WorkPackageProcessor.

virtual void BiometricEvaluation::MPI::RecordProcessor::processRecord (const std::string & *key*) [pure virtual]

Method implemented by child classes to perform an action using each record from the Record Store.

The source RecordStore must be accessible to the the implementation as the value for each key is not included.

Parameters

in	*key*	The key associated with the record that is to be processed.

Exceptions

Error::Exception	An error occurred processing the record: Missing record, input/output error, or memory allocation.

virtual void BiometricEvaluation::MPI::RecordProcessor::processRecord (const std::string & *key*, const Memory::uint8Array & *value*) [pure virtual]

Method implemented by child classes to perform an action using each record from the Record Store.
Parameters

in	*key*	The key associated with the record that is to be processed.
in	*value*	The data from the record that is to be processed.

Exceptions

Error::Exception	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.

void BiometricEvaluation::MPI::RecordProcessor::processWorkPackage (MPI::WorkPackage & *workPackage*) [virtual]

Process the data contents of the work package. This method is part of the worker personality.
Parameters

in	*workPackage*	The work package.

Exceptions

Error::Exception	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.

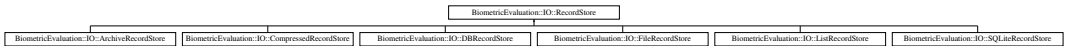
Implements BiometricEvaluation::MPI::WorkPackageProcessor.

H.96 BiometricEvaluation::IO::RecordStore Class Reference

A class to represent a data storage mechanism.

`#include <be_io_recordstore.h>`

Inheritance diagram for BiometricEvaluation::IO::RecordStore:



Public Types

- enum Kind {
  Kind::BerkeleyDB, Kind::Archive, Kind::File, Kind::SQLite,
  Kind::Compressed, Kind::List, Kind::Default = BerkeleyDB }
- using **iterator** = IO::RecordStoreIterator
- using **const_iterator** = const IO::RecordStoreIterator

Public Member Functions

- std::string getDescription () const
- unsigned int getCount () const
- std::string getPathname () const
- virtual void move (const std::string &pathname)

  Move the RecordStore.
- virtual void changeDescription (const std::string &description)
- virtual uint64_t getSpaceUsed () const

  Obtain real storage utilization.
- virtual void sync () const
- virtual void insert (const std::string &key, const void *const data, const uint64_t size)=0
- virtual void insert (const std::string &key, const Memory::uint8Array &data)
- virtual void remove (const std::string &key)=0
- virtual uint64_t read (const std::string &key, void *const data) const =0
- virtual uint64_t read (const std::string &key, Memory::uint8Array &data) const

  Read a complete record from a store.
- virtual void replace (const std::string &key, const void *const data, const uint64_t size)=0
- virtual void replace (const std::string &key, const Memory::uint8Array &data)
- virtual uint64_t length (const std::string &key) const =0
- virtual void flush (const std::string &key) const =0
- virtual uint64_t sequence (std::string &key, void *const data=nullptr, int cursor=BE_RECSTORE_SEQ_NEXT)=0

  Sequence through a RecordStore, returning the key/data pairs.
- virtual uint64_t sequence (std::string &key, Memory::uint8Array &data, int cursor=BE_RECSTORE_SEQ_NEXT)

  Sequence through a RecordStore, returning the key/data pairs.
- virtual void setCursorAtKey (const std::string &key)=0
- virtual bool containsKey (const std::string &key) const

  Determines whether the RecordStore contains an element with the specified key.
- virtual iterator begin () noexcept
- virtual iterator end () noexcept

Static Public Member Functions

- static std::shared_ptr
  < RecordStore > openRecordStore (const std::string &pathname, uint8_t mode=READWRITE)

  Open an existing RecordStore and return a managed pointer to the the object representing that store.
- static std::shared_ptr
  < RecordStore > createRecordStore (const std::string &pathname, const std::string &description, const Kind &kind)

  Create a new RecordStore and return a managed pointer to the the object representing that store.
- static void removeRecordStore (const std::string &pathname)
- static void mergeRecordStores (const std::string &mergePathname, const std::string &description, const RecordStore::Kind &kind, const std::vector< std::string > &pathnames)

  Create a new RecordStore that contains the contents of several other RecordStores.

Static Public Attributes

- static const std::string INVALIDKEYCHARS
- static const char KEY_SEGMENT_SEPARATOR = '&'
- static const uint64_t KEY_SEGMENT_START = 1
- static const std::string CONTROLFILENAME
- static const std::string DESCRIPTIONPROPERTY
- static const std::string COUNTPROPERTY
- static const std::string TYPEPROPERTY
- static const std::string RSREADONLYERROR
- static const int BE_RECSTORE_SEQ_START = 1
- static const int BE_RECSTORE_SEQ_NEXT = 2

Protected Member Functions

- RecordStore (const std::string &pathname, const std::string &description, const Kind &kind)
- RecordStore (const std::string &pathname, uint8_t mode=READWRITE)
- uint8_t **getMode** () const
- std::string **canonicalName** (const std::string &name) const
- int **getCursor** () const
- void **setCursor** (int cursor)
- bool **validateKeyString** (const std::string &key) const
- void setProperties (const std::shared_ptr< IO::Properties > properties)

  Replace existing Properties in RecordStore control file.
- std::shared_ptr< IO::Properties > getProperties () const

  Obtain a copy of the Properties object.

Static Protected Member Functions

- static std::string genKeySegName (const std::string &key, const uint64_t segnum)

  Generate key segment names.

H.96.1 Detailed Description

A class to represent a data storage mechanism.

A RecordStore is an abstraction that associates keys with a specific record. Implementations of this abstraction can store the records in any format supported by the operating system, such as files or databases, rooted in the file system.

Certain characters are prohibited in the key string. See IO::RecordStore::INVALIDKEYCHARS. A key string cannot begin with the space character.

See also

IO::ArchiveRecordStore, IO::DBRecordStore, IO::FileRecordStore.

H.96.2 Member Enumeration Documentation

enum BiometricEvaluation::IO::RecordStore::Kind `[strong]`

Possible types of RecordStore

Enumerator

BerkeleyDB DBRecordStore

Archive ArchiveRecordStore

File FileRecordStore

SQLite SQLiteRecordStore

Compressed CompressedRecordStore

List ListRecordStore

Default "Default" RecordStore kind

H.96.3 Constructor & Destructor Documentation

BiometricEvaluation::IO::RecordStore::RecordStore (const std::string & *pathname*, const std::string & *description*, const Kind & *kind*) `[protected]`

Constructor to create a new RecordStore.

Parameters

in	*pathname*	The pathname where the RecordStore is to be created.
in	*description*	The text used to describe the store.
in	*kind*	The kind of RecordStore.

Exceptions

Error::ObjectExists	The store was previously created, or the directory where it would be created exists.
Error::StrategyError	An error occurred when using the underlying storage system.

BiometricEvaluation::IO::RecordStore::RecordStore (const std::string & *pathname*, uint8_t *mode* = *READWRITE*) `[protected]`

Constructor to open an existing RecordStore.

Parameters

in	*pathname*	The pathname where the RecordStore is to be created.
in	*mode*	The type of access a client of this RecordStore has.

Exceptions

Error::ObjectDoesNotExist	The RecordStore does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

H.96.4 Member Function Documentation

virtual iterator BiometricEvaluation::IO::RecordStore::begin ( ) `[virtual], [noexcept]`

Returns

Iterator to the first record.

virtual void BiometricEvaluation::IO::RecordStore::changeDescription (const std::string & *description*) [virtual]

Change the description of the RecordStore.

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError	An error occurred when using the underlying storage system.

Reimplemented in BiometricEvaluation::IO::SQLiteRecordStore.

virtual bool BiometricEvaluation::IO::RecordStore::containsKey (const std::string & *key*) const [virtual]

Determines whether the RecordStore contains an element with the specified key.

Parameters

key	The key to locate.

Returns

True if the RecordStore contains an element with the key, false otherwise.

static std::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::createRecordStore (const std::string & *pathname*, const std::string & *description*, const Kind & *kind*) [static]

Create a new RecordStore and return a managed pointer to the the object representing that store.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

in	*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.

virtual iterator BiometricEvaluation::IO::RecordStore::end ( ) [virtual], [noexcept]

Returns

Iterator past the last record.

virtual void BiometricEvaluation::IO::RecordStore::flush (const std::string & *key*) const [pure virtual]

Commit the record's data to storage.

Parameters

in	*key*	The key of the record to be flushed.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implemented in BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::ListRecordStore, BiometricEvaluation::IO::FileRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

static std::string BiometricEvaluation::IO::RecordStore::genKeySegName (const std::string & *key*, const uint64_t *segnum*) [static], [protected]

Generate key segment names.

Parameters

key	Base key name.
segnum	Segment number for key (zero based).

Returns

Key segment name.

unsigned int BiometricEvaluation::IO::RecordStore::getCount ( ) const

Obtain the number of items in the RecordStore.

Returns

The number of items in the RecordStore.

std::string BiometricEvaluation::IO::RecordStore::getDescription ( ) const

Obtain a textual description of the RecordStore.

Returns

The RecordStore's description.

std::string BiometricEvaluation::IO::RecordStore::getPathname ( ) const

Return the path name of the RecordStore.

Returns

Where in the file system the RecordStore is located.

std::shared_ptr<IO::Properties> BiometricEvaluation::IO::RecordStore::getProperties ( ) const [protected]

Obtain a copy of the Properties object.
RecordStore core properties will be excluded.

Returns

Shared pointer to Properties object that may be modified.

virtual uint64_t BiometricEvaluation::IO::RecordStore::getSpaceUsed ( ) const [virtual]

Obtain real storage utilization.
The amount of disk space used, for example. This is the actual space allocated by the underlying storage mechanism, in bytes.

Returns

The amount of backing storage used by the RecordStore.

Exceptions

Error::StrategyError	An error occurred when using the underlying storage system.

Reimplemented in BiometricEvaluation::IO::ListRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual void BiometricEvaluation::IO::RecordStore::insert (const std::string & *key*, const void *const *data*, const uint64_t *size*) [pure virtual]

Insert a record into the store.
Parameters

in	*key*	The key of the record to be inserted.
in	*data*	The data for the record.
in	*size*	The size, in bytes, of the record.

Exceptions

Error::ObjectExists	A record with the given key is already present.
Error::StrategyError	An error occurred when using the underlying storage system.

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual void BiometricEvaluation::IO::RecordStore::insert (const std::string & *key*, const Memory::uint8Array & *data*) [virtual]

Insert a record into the store.

Parameters

in	*key*	The key of the record to be inserted.
in	*data*	The data for the record.

Exceptions

Error::ObjectExists	A record with the given key is already present.
Error::StrategyError	An error occurred when using the underlying storage system.

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.

Implemented in BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

static void BiometricEvaluation::IO::RecordStore::mergeRecordStores (const std::string & *mergePathname*, const std::string & *description*, const RecordStore::Kind & *kind*, const std::vector< std::string > & *pathnames*) [static]

Create a new RecordStore that contains the contents of several other RecordStores.

Parameters

in	*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.

virtual void BiometricEvaluation::IO::RecordStore::move (const std::string & *pathname*) [virtual]

Move the RecordStore.

The RecordStore can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the RecordStore.

Exceptions

Error::StrategyError	An error occurred when using the underlying storage system.

Reimplemented in BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::Archive↩ RecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::ListRecordStore, Biometric↩ Evaluation::IO::FileRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

static std::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::openRecordStore (const std::string & *pathname*, uint8_t *mode* = *READWRITE*) [static]

Open an existing RecordStore and return a managed pointer to the the object representing that store.

Applications can open existing record stores without the need to know what type of RecordStore it is.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

in	*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::ObjectDoesNot↩ Exist	The RecordStore does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

virtual uint64_t BiometricEvaluation::IO::RecordStore::read (const std::string & *key*, void *const *data*) const [pure virtual]

Read a complete record from a store. Applications are responsible for allocating storage for the record's data.

Parameters

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::Compressed↩ RecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, Biometric↩ Evaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual uint64_t BiometricEvaluation::IO::RecordStore::read (const std::string & *key*, Memory::uint8Array & *data*) const [virtual]

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

virtual void BiometricEvaluation::IO::RecordStore::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.

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

static void BiometricEvaluation::IO::RecordStore::removeRecordStore (const std::string & *pathname*) [static]

Remove a RecordStore by deleting all 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.

virtual void BiometricEvaluation::IO::RecordStore::replace (const std::string & *key*, const void *const *data*, const uint64_t *size*) [pure virtual]

Replace a complete record in a store.

Parameters

in	*key*	The key of the record to be replaced.
in	*data*	The data for the record.
in	*size*	The size of data.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual void BiometricEvaluation::IO::RecordStore::replace (const std::string & *key*, const Memory::uint8Array & *data*) [virtual]

Replace a complete record in a RecordStore.

Parameters

in	*key*	The key of the record to be replaced.
in	*data*	The data for the record.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

virtual uint64_t BiometricEvaluation::IO::RecordStore::sequence (std::string & *key*, void *const *const *data* = *nullptr*, int *cursor* = BE_RECSTORE_SEQ_NEXT) [pure virtual]

Sequence through a RecordStore, returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the sequencor to return the next record. The starting point is typically the first record, and is set to that when the RecordStore object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to nullptr to indicate only the key is wanted.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implemented in BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::Archive↩ RecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::ListRecordStore, Biometric↩ Evaluation::IO::FileRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual uint64_t BiometricEvaluation::IO::RecordStore::sequence (std::string & *key*, Memory::uint8Array & *data*, int *cursor* = BE_RECSTORE_SEQ_NEXT) [virtual]

Sequence through a RecordStore, returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the sequencor to return the next record. The starting point is typically the first record, and is set to that when the RecordStore object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

virtual void BiometricEvaluation::IO::RecordStore::setCursorAtKey (const std::string & *key*) [pure virtual]

Set the sequence cursor to an arbitrary position within the RecordStore, starting at key. Key will be the first record returned from the next call to sequence().

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to sequence().

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implemented in BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::Archive↩ RecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::ListRecordStore, Biometric↩ Evaluation::IO::FileRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

void BiometricEvaluation::IO::RecordStore::setProperties (const std::shared_ptr< IO::Properties > *properties*) [protected]

Replace existing Properties in RecordStore control file.

Existing properties will be updated. RecordStore core properties will be ignored.

Parameters

in	*properties*	Shared pointer to Properties object.

Exceptions

Error::StrategyError	RecordStore was opened READONLY.

virtual void BiometricEvaluation::IO::RecordStore::sync ( ) const [virtual]

Synchronize the entire record store to persistent storage.
Exceptions

Error::StrategyError	An error occurred when using the underlying storage system.

Reimplemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::DBRecordStore.

H.96.5 Member Data Documentation

const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_NEXT = 2 [static]

Tell sequence to sequence from current position

const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_START = 1 [static]

Tell sequence() to sequence from beginning

const std::string BiometricEvaluation::IO::RecordStore::CONTROLFILENAME [static]

The name of the control file, a properties list

const std::string BiometricEvaluation::IO::RecordStore::COUNTPROPERTY [static]

Property key for the number of store items

const std::string BiometricEvaluation::IO::RecordStore::DESCRIPTIONPROPERTY [static]

Property key for description of the RecordStore

const std::string BiometricEvaluation::IO::RecordStore::INVALIDKEYCHARS [static]

The set of prohibited characters in a key: '/', '\', '*', '&'

const char BiometricEvaluation::IO::RecordStore::KEY_SEGMENT_SEPARATOR = '&' [static]

Character used to separate key segments

const uint64_t BiometricEvaluation::IO::RecordStore::KEY_SEGMENT_START = 1 [static]

First segment number of a segmented record

const std::string BiometricEvaluation::IO::RecordStore::RSREADONLYERROR [static]

Message for READONLY RecordStore modification

const std::string BiometricEvaluation::IO::RecordStore::TYPEPROPERTY [static]

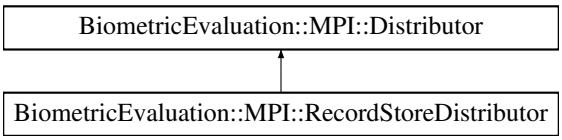
Property key for the type of RecordStore

H.97 BiometricEvaluation::MPI::RecordStoreDistributor Class Reference

An implementation of the Distrbutor abstraction that uses a record store for input to create the work packages.

`#include <be_mpi_recordstoredistributor.h>`

Inheritance diagram for BiometricEvaluation::MPI::RecordStoreDistributor:



Public Member Functions

- RecordStoreDistributor (const std::string &propertiesFileName, const bool includeValues)

  Construct a distributor using the named properties.

Protected Member Functions

- void createWorkPackage (MPI::WorkPackage &workPackage)

  Create a work package for distribution.

H.97.1 Detailed Description

An implementation of the Distrbutor abstraction that uses a record store for input to create the work packages.

H.97.2 Constructor & Destructor Documentation

BiometricEvaluation::MPI::RecordStoreDistributor::RecordStoreDistributor (const std::string & *propertiesFileName*, const bool *includeValues*)

Construct a distributor using the named properties.

The distributor object is based on the properties given in the file. The name of the input record store must be one of the properties.

The work package sent to Receivers can contain either RecordStore keys, or key/value pairs.

Note

The size of a single value item is limited to 2^32 octets. If the size of the value item is larger, behavior is undefined.

Parameters

in	*propertiesFile↩ Name*	The file containing the properties.
in	*includeValues*	true if both the key and value items are included in the work package, false otherwise.

Exceptions

Error::Exception	An error occurred, typically due to missing or invalid properties.

See also

MPI::Distributor
MPI::RecordProcessor
MPI::RecordStoreResources

H.97.3 Member Function Documentation

void BiometricEvaluation::MPI::RecordStoreDistributor::createWorkPackage (MPI::WorkPackage & *workPackage*) [protected], [virtual]

Create a work package for distribution.

Implementations of this class create a work package to encapsulate the specific data type that is to be distributed.

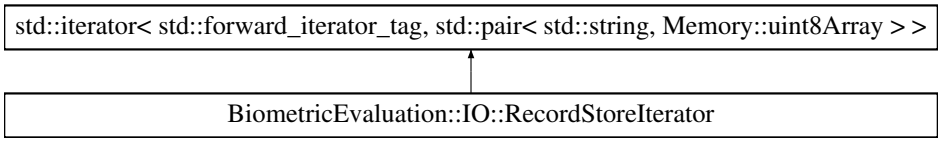
Implements BiometricEvaluation::MPI::Distributor.

H.98 BiometricEvaluation::IO::RecordStoreIterator Class Reference

Generic ForwardIterator for all RecordStores.

`#include <be_io_recordstoreiterator.h>`

Inheritance diagram for BiometricEvaluation::IO::RecordStoreIterator:



Public Member Functions

- RecordStoreIterator ()

  Default constructor.
- RecordStoreIterator (IO::RecordStore ∗recordStore, bool atEnd=false)

  Constructor.
- RecordStoreIterator (const RecordStoreIterator &rhs)=default
- RecordStoreIterator (RecordStoreIterator &&rvalue)=default
- virtual ∼RecordStoreIterator ()=default
- reference operator∗ ()
- pointer operator-> ()
- RecordStoreIterator operator++ ()

- RecordStoreIterator operator++ (int postfix)
- RecordStoreIterator operator+= (difference_type rhs)

  Advance a variable number of arguments.
- RecordStoreIterator operator+ (difference_type rhs)

  Advance a variable number of arguments.
- bool operator== (const RecordStoreIterator &rhs)

  Equivalence operator.
- bool operator!= (const RecordStoreIterator &rhs)

  Non-equivalence operator.
- RecordStoreIterator & **operator=** (RecordStoreIterator &rhs)=default
- RecordStoreIterator & operator= (RecordStoreIterator &&rhs)=default

H.98.1 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().

H.98.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator ( )

Default constructor.
Creates ”end” iterator.

Note

Satisfies DefaultConstructible requirement.

BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator (IO::RecordStore * *recordStore*, bool *atEnd* = `false`)

Constructor.
Parameters

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.

BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator (const RecordStoreIterator & *rhs*) `[default]`

Default copy constructor

BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator (RecordStoreIterator && *rvalue*) `[default]`

Default move constructor

virtual BiometricEvaluation::IO::RecordStoreIterator::~RecordStoreIterator ( ) `[virtual], [default]`

Default destructor

H.98.3 Member Function Documentation

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.

reference BiometricEvaluation::IO::RecordStoreIterator::operator* ( )

Returns

Reference to a key/value pair.

RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator+ (difference_type *rhs*)

Advance a variable number of arguments.
Parameters

rhs	Number of objects to advance (1 or more).

Returns

Self after advancing rhs objects.

RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator++ ( )

Returns

Self after advancing.

RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator++ (int *postfix*)

Returns

Copy of self before advancing.

RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator+= (difference_type *rhs*)

Advance a variable number of arguments.

Parameters

rhs	Number of objects to advance (1 or more).

Returns

Self after advancing rhs objects.

pointer BiometricEvaluation::IO::RecordStoreIterator::operator-> ( )

Returns

A dereferenced key/value pair.

RecordStoreIterator& BiometricEvaluation::IO::RecordStoreIterator::operator= (RecordStoreIterator && *rhs*) `[default]`

Default move assignment operator

bool BiometricEvaluation::IO::RecordStoreIterator::operator== (const RecordStoreIterator & *rhs*)

Equivalence operator.
Parameters

rhs	Reference to RecordStoreIterator being compared.

Returns

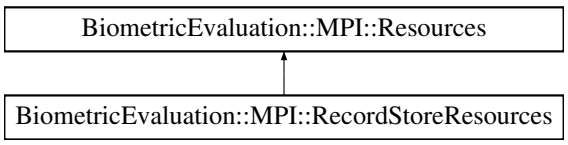
Whether or not this is equivalent to rhs.

H.99 BiometricEvaluation::MPI::RecordStoreResources Class Reference

A class to represent a set of resources needed by an MPI program using a RecordStore for input.

`#include <be_mpi_recordstoreresources.h>`

Inheritance diagram for BiometricEvaluation::MPI::RecordStoreResources:



Public Member Functions

- RecordStoreResources (const std::string &propertiesFileName)

  Constructor taking the name of the properties file with the resource names.
- uint32_t **getChunkSize** () const
- bool haveRecordStore () const

  Indicator that a record store has been opened.
- std::shared_ptr< IO::RecordStore > getRecordStore () const

  Return the RecordStore named in the property set.

Static Public Member Functions

- static std::vector< std::string > getRequiredProperties ()

  Obtain the required properties as strings.
- static std::vector< std::string > getOptionalProperties ()

  Obtain the list of optional properties.

Static Public Attributes

- static const std::string INPUTRSPROPERTY

  The property string "Input Record Store"; required.
- static const std::string CHUNKSIZEPROPERTY

  The property string "Chunk Size"; required.

H.99.1 Detailed Description

A class to represent a set of resources needed by an MPI program using a RecordStore for input.

Resources are opened based on the property when appropriate. The input record store need not be accessible. Applications should call haveRecordStore() to check whether the record store has been opened.

H.99.2 Constructor & Destructor Documentation

BiometricEvaluation::MPI::RecordStoreResources::RecordStoreResources (const std::string & *propertiesFileName*)

Constructor taking the name of the properties file with the resource names.

Exceptions

Error::FileError	The resources file could not be read.
Error::ObjectDoesNotExist	A required property does not exist.
Error::Exception	Some other error occurred.

H.99.3 Member Function Documentation

static std::vector<std::string> BiometricEvaluation::MPI::RecordStoreResources::getOptionalProperties ( ) `[static]`

Obtain the list of optional properties.

Returns

A set of optional property strings.

std::shared_ptr<IO::RecordStore> BiometricEvaluation::MPI::RecordStoreResources::getRecordStore ( ) const

Return the RecordStore named in the property set.

Returns

A shared pointer to the record store.

static std::vector<std::string> BiometricEvaluation::MPI::RecordStoreResources::getRequiredProperties ( ) [static]

Obtain the required properties as strings.

Returns

The set of required properties.

bool BiometricEvaluation::MPI::RecordStoreResources::haveRecordStore ( ) const

Indicator that a record store has been opened.

Returns

true if input record store is opened, false otherwise.

H.100 BiometricEvaluation::Image::Resolution Struct Reference

A structure to represent the resolution of an image.

`#include <be_image.h>`

Public Types

- enum Units { Units::NA = 0, Units::PPI = 1, Units::PPMM = 2, Units::PPCM = 3 }

  Possible representations of the units in a Resolution struct.

Public Member Functions

- Resolution (const double xRes=0.0, const double yRes=0.0, const Units units=Units::PPI)

  Create a Resolution struct.

Public Attributes

- double xRes
- double yRes
- Units units

H.100.1 Detailed Description

A structure to represent the resolution of an image.

H.100.2 Member Enumeration Documentation

enum BiometricEvaluation::Image::Resolution::Units [strong]

Possible representations of the units in a Resolution struct.

Enumerator

NA Not-applicable: unknown, or otherwise

PPI Pixels per inch

PPMM Pixels per millimeter

PPCM Pixels per centimeter

H.100.3 Constructor & Destructor Documentation

BiometricEvaluation::Image::Resolution::Resolution (const double *xRes* = `0.0`, const double *yRes* = `0.0`, const Units *units* = Units::PPI)

Create a Resolution struct.

Parameters

in	*xRes*	Resolution along the X-axis
in	*yRes*	Resolution along the Y-axis
in	*units*	Units in which xRes and yRes are represented

H.100.4 Member Data Documentation

Units BiometricEvaluation::Image::Resolution::units

Units in which xRes and yRes are represented

double BiometricEvaluation::Image::Resolution::xRes

Resolution along the X-axis

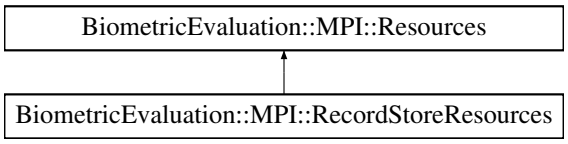
double BiometricEvaluation::Image::Resolution::yRes

Resolution along the Y-axis

H.101 BiometricEvaluation::MPI::Resources Class Reference

`#include <be_mpi_resources.h>`

Inheritance diagram for BiometricEvaluation::MPI::Resources:



Public Member Functions

- Resources (const std::string &propertiesFileName)

  Constructor taking the name of the properties file describing the resources.
- std::string getPropertiesFileName () const

  Obtain the name of the file used to construct this object.
- std::string getLogsheetURL () const

  Obtain the Uniform Resource Locator for the IO:Logsheet object.
- int **getRank** () const
- int **getNumTasks** () const
- int **getWorkersPerNode** () const

Static Public Member Functions

- static std::vector< std::string > getRequiredProperties ()

  Obtain the list of required properties.
- static std::vector< std::string > getOptionalProperties ()

  Obtain the list of optional properties.

Static Public Attributes

- static const std::string WORKERSPERNODEPROPERTY

  The property string "Workers Per Node"; required.
- static const std::string LOGSHEETURLPROPERTY

  The property string "Logsheet URL"; optional.

H.101.1 Detailed Description

A class to represent a set of resources needed by an MPI program. The resources are based on a properties file as well as some dynamic information, such as MPI rank and process ID.

H.101.2 Constructor & Destructor Documentation

BiometricEvaluation::MPI::Resources::Resources (const std::string & *propertiesFileName*)

Constructor taking the name of the properties file describing the resources.
Parameters

in	*propertiesFileName*	The name of the file containing the Properties.

Exceptions

Error::FileError	The resources file could not be read.
Error::ObjectDoesNotExist	A required property does not exist.
Error::Exception	Some other error occurred.

H.101.3 Member Function Documentation

std::string BiometricEvaluation::MPI::Resources::getLogsheetURL ( ) const

Obtain the Uniform Resource Locator for the IO:Logsheet object.
This string my be empty, indicating that there is no Logsheet URL in the Properties file.

Returns

The Logsheet URL.

static std::vector<std::string> BiometricEvaluation::MPI::Resources::getOptionalProperties ( ) `[static]`

Obtain the list of optional properties.

Returns

A set of optional property strings.

std::string BiometricEvaluation::MPI::Resources::getPropertiesFileName ( ) const

Obtain the name of the file used to construct this object.

Returns

The name of the properties file.

static std::vector<std::string> BiometricEvaluation::MPI::Resources::getRequiredProperties ( )
`[static]`

Obtain the list of required properties.

Returns

A set of required property strings.

H.102 BiometricEvaluation::Feature::RidgeCountItem Struct Reference

Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number.

`#include <be_feature_minutiae.h>`

Public Member Functions

- RidgeCountItem (RidgeCountExtractionMethod extraction_method, int index_one, int index_two, int count=0)

  Create a RidgeCountItem struct.

Public Attributes

- RidgeCountExtractionMethod **extraction_method**
- int **index_one**
- int **index_two**
- int **count**

H.102.1 Detailed Description

Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number.

H.103 BiometricEvaluation::MPI::Runtime Class Reference

Runtime support for the startup/shutdown of MPI jobs.

`#include <be_mpi_runtime.h>`

Public Member Functions

- Runtime (int &argc, char **&argv)

  Construct the runtime environment for the processes making up the MPI job.
- void start (BiometricEvaluation::MPI::Distributor &distributor, BiometricEvaluation::MPI::Receiver &receiver)

  Startup the runtime environment for the MPI job.
- void shutdown ()

  Shutdown the runtime environment for the MPI job.
- void abort (int errcode)

  Abort the runtime the MPI job.

H.103.1 Detailed Description

Runtime support for the startup/shutdown of MPI jobs.

This class provides methods that are used by applications to start and shutdown the MPI job. Each job consists of a single distributor of work, and 1..n receivers of work which then distribute the work packages to child processes to take action on the work package.

H.103.2 Constructor & Destructor Documentation

BiometricEvaluation::MPI::Runtime::Runtime (int & *argc*, char ∗∗& *argv*)

Construct the runtime environment for the processes making up the MPI job.

Parameters

in	*argc*	The argument count, taken from the command line passed to main().
in	*argv*	The argument vector, taken from the command line passed to main().

H.103.3 Member Function Documentation

void BiometricEvaluation::MPI::Runtime::abort (int *errcode*)

Abort the runtime the MPI job.

This method will cause the MPI job to terminate immediately. All processes will end without the opportunity to save.

Parameters

in	*errocode*	The error code to return to the MPI runtime.

void BiometricEvaluation::MPI::Runtime::shutdown ( )

Shutdown the runtime environment for the MPI job.

This method must be called in order for the MPI runtime to cleanly exit.

void BiometricEvaluation::MPI::Runtime::start (BiometricEvaluation::MPI::Distributor & *distributor*, BiometricEvaluation::MPI::Receiver & *receiver*)

Startup the runtime environment for the MPI job.

Parameters

in	*distributor*	The Distributor object that will form the basis of the first MPI task.
in	*receiver*	The Receiver object which will form the basis of MPI tasks 1..n.

H.104 BiometricEvaluation::Process::Semaphore Class Reference

Represent a semaphore that can be used for interprocess communication.

`#include <be_process_semaphore.h>`

Public Member Functions

- Semaphore (const std::string &name, const mode_t mode, const int value, const bool exclusive=false)

  Create a new named sempahore.
- Semaphore (const std::string &name)

Open an existing named sempahore.

- bool wait (const bool interruptible)

  Wait indefinitely for the semaphore to unblock.

- bool trywait (const bool interruptible)

  Attempt to obtain the semaphore without blocking.

- bool timedwait (const uint64_t interval, const bool interruptible)

  Attempt to obtain the semaphore while blocking for at most the specified time interval.

- void post ()

  Post (increment) to the semaphore.

H.104.1 Detailed Description

Represent a semaphore that can be used for interprocess communication.

Semaphores are shared counters with mutually exclusive modification properties. A counter value greater than zero means that a resource represented by the semaphore is available. A typical use is to grant exclusive access to a resource by allowing the counter to be valued at zero or one; this is known as a binary semaphore.

Note

The counter value is not exposed to clients of the object.

H.104.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::Semaphore::Semaphore (const std::string & *name*, const mode_t *mode*, const int *value*, const bool *exclusive* = `false`)

Create a new named sempahore.

Parameters

in	*name*	The name of the semaphore, which must obey the syntax documented for the sem_open(2) call. If the semaphore already exists in the name space, construction will fail unless the exclusive flag is true. In that case, the existing semaphore will be removed.
in	*mode*	The permission mode of the semaphore.
in	*value*	The initial value of the semaphore.
in	*exclusive*	The semaphore is created only when it doesn't already exist.

Exceptions

Error::ObjectExists	The semaphore already exists with the given name.
Error::StrategyError	An error occurred when creating the semaphore.

BiometricEvaluation::Process::Semaphore::Semaphore (const std::string & *name*)

Open an existing named sempahore.

Parameters

in	*name*	The name of the semaphore, which must obey the syntax documented for the sem_open(2) call.

H.104.3 Member Function Documentation

void BiometricEvaluation::Process::Semaphore::post ( )

Post (increment) to the semaphore.

Exceptions

Error::ObjectDoesNot↩ Exist	The semaphore is no longer valid.
Error::StrategyError	System error obtaining the semaphore.

bool BiometricEvaluation::Process::Semaphore::timedwait (const uint64_t *interval,* const bool *interruptible*)

Attempt to obtain the semaphore while blocking for at most the specified time interval.
Parameters

in	*interval*	The max time to wait, in microseconds.
in	*interruptible*	true if the function should return if waiting was interrupted, false otherwise.

Returns

true if the semaphore was obtained; false if not.

Exceptions

Error::ObjectDoesNot↩ Exist	The semaphore is no longer valid.
Error::NotImplemented	Function is not implemented on the system. Applications should then call wait() or trywait().
Error::StrategyError	System error obtaining the semaphore.

bool BiometricEvaluation::Process::Semaphore::trywait (const bool *interruptible*)

Attempt to obtain the semaphore without blocking.
Parameters

in	*interruptible*	true if the function should return if waiting was interrupted, false otherwise.

Returns

true if the semaphore was obtained; false if not.

Exceptions

Error::ObjectDoesNot↩ Exist	The semaphore is no longer valid.
Error::StrategyError	System error obtaining the semaphore.

bool BiometricEvaluation::Process::Semaphore::wait (const bool *interruptible*)

Wait indefinitely for the semaphore to unblock.
Parameters

in	*interruptible*	true if the function should return if waiting was interrupted, false otherwise.

Returns

true if the semaphore was obtained; false if not.

Exceptions

Error::ObjectDoesNot↩ Exist	The semaphore is no longer valid.
Error::StrategyError	System error obtaining the semaphore.

H.105 BiometricEvaluation::Error::SignalManager Class Reference

A SignalManager object is used to handle signals that come from the operating system.

`#include <be_error_signal_manager.h>`

Public Member Functions

- SignalManager ()
- SignalManager (const sigset_t signalSet)
- void setSignalSet (const sigset_t signalSet)
- void clearSignalSet ()
- void setDefaultSignalSet ()
- bool sigHandled ()
- void start ()
- void stop ()
- void setSigHandled ()
- void clearSigHandled ()

Static Public Attributes

- static bool _canSigJump
- static sigjmp_buf _sigJumpBuf

H.105.1 Detailed Description

A SignalManager object is used to handle signals that come from the operating system.

Applications typically do not invoke most methods of a SignalManager, except the setSignalSet(), set↩DefaultSignalSet(), and sigHandled(). An application wishing to just catch memory errors can simply construct a SignalManager object, and invoke sigHandled() at the end of the signal block to detect whether a signal was handled.

The BEGIN_SIGNAL_BLOCK macro sets up the jump block and tells the SignalManager object to start handling signals. Applications can call either setSignalSet() or setDefaultSignalSet() before invoking these macros to indicate which signals are to be handled.

The END_SIGNAL_BLOCK() macro clears the signal set, so from that point forward application code signals will be handled in the system's default manner until another signal block is created.

The ABORT_SIGNAL_MANAGER() macro also disables the watchdog timer but does not create the code point destination for the jump point. This macro should be used to disable a SignalManager object when the application is no longer interested in the signal handling.

Attention

The BEGIN_SIGNAL_BLOCK() macro must be paired with either the END_SIGNAL_BLOCK() macro or ABORT_SIGNAL_MANAGER() macro. Failure to do so may result in undefined behavior as an active SignalManager may be invoked, forcing a jump into an incompletely initialized function.

A SignalManager is passive (i.e. no signal handlers are installed) until that start() method is called, and becomes passive when stop() is invoked. The signals that are to be handled by the object are maitained as state, and the set of signals can be changed at any time, but are not in effect until start() is called.

Attention

The start(), stop(), setSigHandled() and clearSigHandled() methods are not meant to be used directly by applications, which should use the BEGIN_SIGNAL_BLOCK()/END_SIGNAL_BLOCK() macro pair.

H.105.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::SignalManager::SignalManager ( )

Construct a new SignalManager object with the default signal handling: SIGSEGV and SIGBUS.

Exceptions

Error::StrategyError	Could not register the signal handler.

BiometricEvaluation::Error::SignalManager::SignalManager (const sigset_t *signalSet*)

Construct a new SignalManager object with the specified signal handling, no defaults.

Parameters

signalSet	(in) The signal set; see sigaction(2), sigemptyset(3) and sigaddset(3).

Exceptions

Error::ParameterError	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

H.105.3 Member Function Documentation

void BiometricEvaluation::Error::SignalManager::clearSigHandled ( )

Clear the indication that a signal was handled.

void BiometricEvaluation::Error::SignalManager::clearSignalSet ( )

Clear all signal handling.

void BiometricEvaluation::Error::SignalManager::setDefaultSignalSet ( )

Set the default signals this object will manage: SIGSEGV and SIGBUS.

void BiometricEvaluation::Error::SignalManager::setSigHandled ( )

Set a flag to indicate a signal was handled.

void BiometricEvaluation::Error::SignalManager::setSignalSet (const sigset_t *signalSet*)

Set the signals this object will manage.

Parameters

signalSet	(in) The signal set; see sigaction(2), sigemptyset(3) and sigaddset(3).

Exceptions

Error::ParameterError	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

bool BiometricEvaluation::Error::SignalManager::sigHandled ( )

Indicate whether a signal was handled.

Returns

true if a signal was handled, false otherwise.

void BiometricEvaluation::Error::SignalManager::start ( )

Start handling signals of the current signal set.

Exceptions

Error::StrategyError	Could not register the signal handler.

Note

If an application invokes start() without setting up a signal jump block, behavior is undefined, and can result in an infinite loop if further processing causes a signal to be raised.

void BiometricEvaluation::Error::SignalManager::stop ( )

Stop handling signals of the current signal set.

Exceptions

Error::StrategyError	Could not register the signal handler.

H.105.4 Member Data Documentation

bool BiometricEvaluation::Error::SignalManager::_canSigJump `[static]`

Flag indicating can jump after handling a signal.

Note

Should not be directly used by applications.

sigjmp_buf BiometricEvaluation::Error::SignalManager::_sigJumpBuf `[static]`

The jump buffer used by the signal handler.

Note

Should not be directly used by applications.

H.106 BiometricEvaluation::Image::Size Struct Reference

A structure to represent the size of an image, in pixels.

`#include <be_image.h>`

Public Member Functions

- Size (const uint32_t xSize=0, const uint32_t ySize=0)

  Create a Size struct.

Public Attributes

- uint32_t xSize
- uint32_t ySize

H.106.1 Detailed Description

A structure to represent the size of an image, in pixels.

H.106.2 Constructor & Destructor Documentation

BiometricEvaluation::Image::Size::Size (const uint32_t *xSize = 0*, const uint32_t *ySize = 0*)

Create a Size struct.

Parameters

in	*xSize*	Number of pixels on the X-axis
in	*ySize*	Number of pixels on the Y-axis

H.106.3 Member Data Documentation

uint32_t BiometricEvaluation::Image::Size::xSize

Number of pixels on the X-axis

uint32_t BiometricEvaluation::Image::Size::ySize

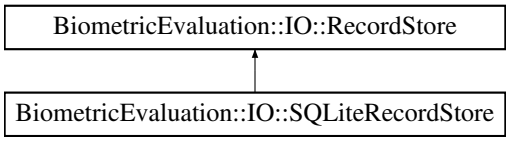
Number of pixels on the Y-axis

H.107 BiometricEvaluation::IO::SQLiteRecordStore Class Reference

A RecordStore implementation using a SQLite database as the underlying record storage system.

`#include <be_io_sqliterecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::SQLiteRecordStore:



Public Member Functions

- **SQLiteRecordStore** (const std::string &pathname, const std::string &description)
- **SQLiteRecordStore** (const std::string &pathname, uint8_t mode=READWRITE)
- void move (const std::string &pathname)

  Move the RecordStore.
- void changeDescription (const std::string &description)
- uint64_t getSpaceUsed () const

  Obtain real storage utilization.
- void insert (const std::string &key, const void *const data, const uint64_t size)
- void remove (const std::string &key)
- uint64_t read (const std::string &key, void *const data) const
- void replace (const std::string &key, const void *const data, const uint64_t size)
- uint64_t length (const std::string &key) const
- void flush (const std::string &key) const
- uint64_t sequence (std::string &key, void *const data=nullptr, int cursor=BE_RECSTORE_SEQ_NEXT)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (const std::string &key)
- **SQLiteRecordStore** (const SQLiteRecordStore &)=delete
- SQLiteRecordStore & **operator=** (const SQLiteRecordStore &)=delete

Protected Member Functions

- void sqliteError (int32_t errorNumber) const

  Convert an SQLite error into a StrategyError.
- void createStructure ()

  Create the tables needed to store key->value pairs in SQLite.
- bool validateKeyValueTable (const std::string &table)

  Confirm that a key->value table exists with the proper schema.
- void createKeyValueTable (const std::string &table)

  Create a tables needed to store key->value pairs in SQLite.
- bool validateSchema ()

  Confirm that the schema of the opened SQLite database is compatible.
- uint64_t readSegments (const std::string &key, void *const data) const

  Select a row from the RecordStore.
- void cleanup ()

  Perform SQLite cleanup routines.

Additional Inherited Members

H.107.1 Detailed Description

A RecordStore implementation using a SQLite database as the underlying record storage system.

H.107.2 Member Function Documentation

void BiometricEvaluation::IO::SQLiteRecordStore::changeDescription (const std::string & *description*) `[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.

Reimplemented from BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::SQLiteRecordStore::cleanup ( ) [protected]

Perform SQLite cleanup routines.

- Finalize the sequencer statement
- Close the SQLite database handle

Exceptions

Error::StrategyError	Bad return code from SQLite during cleanup.

void BiometricEvaluation::IO::SQLiteRecordStore::createKeyValueTable (const std::string & *table*) [protected]

Create a tables needed to store key->value pairs in SQLite.
Parameters

table	Name of the table to create.

Exceptions

Error::StrategyError	Error executing SQL commands.

void BiometricEvaluation::IO::SQLiteRecordStore::createStructure ( ) [protected]

Create the tables needed to store key->value pairs in SQLite.
Exceptions

Error::StrategyError	Error executing SQL commands.

void BiometricEvaluation::IO::SQLiteRecordStore::flush (const std::string & *key*) const [virtual]

Commit the record's data to storage.
Parameters

in	*key*	The key of the record to be flushed.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.

Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::SQLiteRecordStore::getSpaceUsed ( ) const [virtual]

Obtain real storage utilization.

The amount of disk space used, for example. This is the actual space allocated by the underlying storage mechanism, in bytes.

Returns

The amount of backing storage used by the RecordStore.

Exceptions

Error::StrategyError	An error occurred when using the underlying storage system.

Reimplemented from BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::SQLiteRecordStore::insert (const std::string & *key*, const void *const *data*, const uint64_t *size*) [virtual]

Insert a record into the store.

Parameters

in	*key*	The key of the record to be inserted.
in	*data*	The data for the record.
in	*size*	The size, in bytes, of the record.

Exceptions

Error::ObjectExists	A record with the given key is already present.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::SQLiteRecordStore::length (const std::string & *key*) const [virtual]

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.

Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::SQLiteRecordStore::move (const std::string & *pathname*) [virtual]

Move the RecordStore.

The RecordStore can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the RecordStore.

Exceptions

Error::StrategyError	An error occurred when using the underlying storage system.

Reimplemented from BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::SQLiteRecordStore::read (const std::string & *key*, void *const *const *data*) const [virtual]

Read a complete record from a store. Applications are responsible for allocating storage for the record's data.

Parameters

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::SQLiteRecordStore::readSegments (const std::string & *key*, void *const *const *data*) const [protected]

Select a row from the RecordStore.

Parameters

key	Key of the row to select.
data	If not nullptr, deep copy the record for key into data.

Exceptions

Error::ObjectDoesNotExist	Key does not exist in RecordStore.

Error::StrategyError	Error executing SQL commands.

Returns

Size of key's record.

void BiometricEvaluation::IO::SQLiteRecordStore::remove (const std::string & *key*) [virtual]

Remove a record from the store.
Parameters

in	*key*	The key of the record to be removed.

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::SQLiteRecordStore::replace (const std::string & *key*, const void *const *data*, const uint64_t *size*) [virtual]

Replace a complete record in a store.
Parameters

in	*key*	The key of the record to be replaced.
in	*data*	The data for the record.
in	*size*	The size of data.

Exceptions

Error::ObjectDoesNot↩ Exist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::SQLiteRecordStore::sequence (std::string & *key*, void *const *data* = *nullptr*, int *cursor* = BE_RECSTORE_SEQ_NEXT) [virtual]

Sequence through a RecordStore, returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the sequencor to return the next record. The starting point is typically the first record, and is set to that when the RecordStore object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.
Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to nullptr to indicate only the key is wanted.

in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::SQLiteRecordStore::setCursorAtKey (const std::string & *key*) [virtual]

Set the sequence cursor to an arbitrary position within the RecordStore, starting at key. Key will be the first record returned from the next call to sequence().

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to sequence().

Exceptions

Error::ObjectDoesNotExist	A record for the key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::SQLiteRecordStore::sqliteError (int32_t *errorNumber*) const [protected]

Convert an SQLite error into a StrategyError.

Exceptions

Error::StrategyError	Always thrown with the textual description of the last error condition.

bool BiometricEvaluation::IO::SQLiteRecordStore::validateKeyValueTable (const std::string & *table*) [protected]

Confirm that a key->value table exists with the proper schema.

Parameters

table	Name of the table to check.

Returns

Whether or not the table exists with the proper schema.

Exceptions

Error::StrategyError	Error compiling SQL.

bool BiometricEvaluation::IO::SQLiteRecordStore::validateSchema ( ) [protected]

Confirm that the schema of the opened SQLite database is compatible.

Returns

Whether or not the schema of the opened SQLite database is compatible with this object.

Exceptions

Error::StrategyError	Error compiling SQL.

H.108 BiometricEvaluation::Process::Statistics Class Reference

The Statistics class provides an interface for gathering process statistics, such as memory usage, system time, etc.

`#include <be_process_statistics.h>`

Public Member Functions

- Statistics ()
- Statistics (IO::FileLogCabinet *const logCabinet)
- Statistics (const std::shared_ptr< IO::Logsheet > &logSheet)

  Construct a Statistic object that logs to an existing Logsheet.
- void getCPUTimes (uint64_t *usertime, uint64_t *systemtime)
- void getMemorySizes (uint64_t *vmrss, uint64_t *vmsize, uint64_t *vmpeak, uint64_t *vmdata, uint64_t *vmstack)
- uint32_t getNumThreads ()
- void logStats ()

  Create a snapshot of the current process statistics in the FileLogsheet created in the FileLogCabinet.
- void startAutoLogging (uint64_t interval)

  Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.
- void stopAutoLogging ()

  Stop the automatic logging of process statistics.
- void callStatistics_logStats ()

H.108.1 Detailed Description

The Statistics class provides an interface for gathering process statistics, such as memory usage, system time, etc.

The information gathered by objects of this class are for the current process, and can optionally be logged to a FileLogsheet object contained within the provided FileLogCabinet.

Note

The resolution of a returned value for many methods may not match the resolution allowed by the interface. For example, the operating system my allow for second resolution whereas the interface allows microsecond resolution.

H.108.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::Statistics::Statistics ( )

Constructor with no parameters.

BiometricEvaluation::Process::Statistics::Statistics (IO::FileLogCabinet *const *logCabinet*)

Construct a Statistics object with the associated FileLogCabinet.

Parameters

in	*logCabinet*	The FileLogCabinet obejct where this object will create a FileLogsheet to contain the statistic information for the process.

Exceptions

Error::NotImplemented	Logging is not supported on this OS. This exception can be thrown when any portion of the statistics gathering cannot be completed.
Error::ObjectExists	The FileLogsheet already exists. This exception should rarely, if ever, occur.
Error::StrategyError	Failure to create the FileLogsheet in the cabinet.

BiometricEvaluation::Process::Statistics::Statistics (const std::shared_ptr< IO::Logsheet > & *logSheet*)

Construct a Statistic object that logs to an existing Logsheet.

Parameters

in	*logSheet*	Existing Logsheet that will be appended.

Exceptions

Error::NotImplemented	Logging is not supported on this OS. This exception can be thrown when any portion of the statistics gathering cannot be completed.

H.108.3 Member Function Documentation

void BiometricEvaluation::Process::Statistics::callStatistics_logStats ( )

Helper function in C++ space that has access to this object, and is called from C space by the logging thread. Applications should not call this function.

void BiometricEvaluation::Process::Statistics::getCPUTimes (uint64_t * *usertime*, uint64_t * *systemtime*)

Obtain the total user and system times for the process, in microseconds. Any of the out parameters can be nullptr, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

out	*usertime*	Pointer where to store the total user time.
out	*systemtime*	Pointer where to store the total system time.

Exceptions

Error::StrategyError	An error occurred when obtaining the process statistics from the operating system. The exception information string contains the error reason.
Error::NotImplemented	This method is not implemented on this OS.

void BiometricEvaluation::Process::Statistics::getMemorySizes (uint64_t * *vmrss*, uint64_t * *vmsize*, uint64_t * *vmpeak*, uint64_t * *vmdata*, uint64_t * *vmstack*)

Obtain the current memory set sizes for the process, in kilobytes. Any of the out parameters can be nullptr, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

out	*vmrss*	Pointer where to store the current resident set size.
out	*vmsize*	Pointer where to store the current total virtual memory size.
out	*vmpeak*	Pointer where to store the peak total virtual memory size.
out	*vmdata*	Pointer where to store the current virtual memory data segment size.
out	*vmstack*	Pointer where to store the current virtual memory stack segment size.

Exceptions

Error::StrategyError	An error occurred when obtaining the process statistics from the operating system. The exception information string contains the error reason.
Error::NotImplemented	This method is not implemented on this OS.

uint32_t BiometricEvaluation::Process::Statistics::getNumThreads ( )

Obtain the number of threads composing this process.

Note

This method may not be implemented in all operating systems.

Exceptions

Error::StrategyError	An error occurred when obtaining the process info from the operating system. The exception information string contains the error reason.
Error::NotImplemented	This method is not implemented on this OS.

void BiometricEvaluation::Process::Statistics::logStats ( )

Create a snapshot of the current process statistics in the FileLogsheet created in the FileLogCabinet.

Exceptions

Error::ObjectDoesNot↩ Exist	The FileLogsheet does not exist; this object was not created with FileLog↩ Cabinet object.
Error::StrategyError	An error occurred when writing to the FileLogsheet.
Error::NotImplemented	The statistics gathering is not implemented for this operating system.

void BiometricEvaluation::Process::Statistics::startAutoLogging (uint64_t *interval*)

Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.

Note

It is unrealistic to expect that log entries can be made at a rate of one per microsecond.
If stopAutoLogging() is called very soon after the start, a log entry may not be made.

Parameters

in	*interval*	The gap between logging snapshots, in microseconds.

Exceptions

Error::ObjectDoesNot↩ Exist	The FileLogsheet does not exist; this object was not created with FileLog↩ Cabinet object.
Error::ObjectExists	Autologging is currently invoked.
Error::StrategyError	An error occurred when writing to the FileLogsheet.
Error::NotImplemented	The statistics gathering is not implemented for this operating system.

void BiometricEvaluation::Process::Statistics::stopAutoLogging ( )

Stop the automatic logging of process statistics.
Exceptions

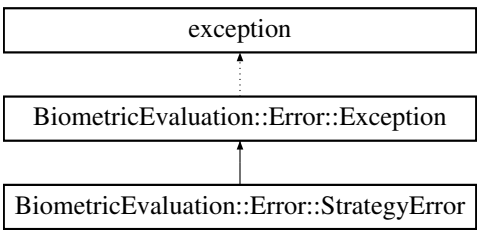
Error::ObjectDoesNot↩ Exist	Not currently autologging.
Error::StrategyError	An error occurred when stopping, most likely because the logging thread died.

H.109 BiometricEvaluation::Error::StrategyError Class Reference

A StrategyError object is thrown when the underlying implementation of this interface encounters an error.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::StrategyError:



Public Member Functions

- StrategyError ()
- StrategyError (std::string info)

H.109.1 Detailed Description

A StrategyError object is thrown when the underlying implementation of this interface encounters an error.

H.109.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::StrategyError::StrategyError ( )

Construct a StrategyError object with the default information string.

BiometricEvaluation::Error::StrategyError::StrategyError (std::string *info*)

Construct a StrategyError object with an information string appended to the default information string.

H.110 BiometricEvaluation::Video::Stream Class Reference

Public Member Functions

- virtual float getFPS ()=0

  Obtain the average frame rate of the video stream.
- virtual uint64_t getFrameCount ()=0

  Obtain the number of frames in the video stream.
- virtual Video::Frame getFrame (uint32_t frameNum)=0

  Obtain a frame from the video stream.
- virtual std::vector< Video::Frame > getFrameSequence (int64_t startTime, int64_t endTime)=0

  Obtain a sequence of frames from the video stream.
- virtual void setFrameScale (float xScale, float yScale)=0

  Set the scaling factors for returned video frames.
- virtual void setFramePixelFormat (const Image::PixelFormat pixelFormat)=0

  Set the pixel format for returned video frames.

H.110.1 Member Function Documentation

virtual float BiometricEvaluation::Video::Stream::getFPS ( ) `[pure virtual]`

Obtain the average frame rate of the video stream.

Returns

The average frame rate. A value of 0 means the frame rate cannot be determined.

virtual Video::Frame BiometricEvaluation::Video::Stream::getFrame (uint32_t *frameNum*) `[pure virtual]`

Obtain a frame from the video stream.

Parameters

frameNum	Frame number, >= 1

Exceptions

Error::ParameterError	frameNum is too large.
Error::StrategyError	No codec available for the video stream or other failure to read the stream.

virtual uint64_t BiometricEvaluation::Video::Stream::getFrameCount ( ) [pure virtual]

Obtain the number of frames in the video stream.

Returns

The number of frames in the stream; will be 0 if unknown.

virtual std::vector<Video::Frame> BiometricEvaluation::Video::Stream::getFrameSequence (int64_t *startTime*, int64_t *endTime*) [pure virtual]

Obtain a sequence of frames from the video stream.

The end time can be greater than the length of the stream, and is not considered an error. Frames up to and including the last will be returned.

Parameters

startTime	Approximate time of the starting frame, microseconds.
endTime	Approximate time of the ending frame, microseconds

Exceptions

Error::StrategyError	No codec available for the video stream or other failure to read the stream.

virtual void BiometricEvaluation::Video::Stream::setFramePixelFormat (const Image::PixelFormat *pixelFormat*) [pure virtual]

Set the pixel format for returned video frames.

Parameters

pixelFormat	The pixel format of all returned frames.

virtual void BiometricEvaluation::Video::Stream::setFrameScale (float *xScale*, float *yScale*) [pure virtual]

Set the scaling factors for returned video frames.

Parameters

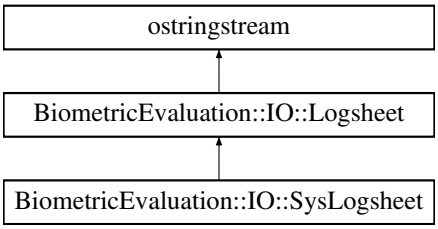
xScale	The scaling factor for frame width.
yScale	The scaling factor for frame height.

H.111 BiometricEvaluation::IO::SysLogsheet Class Reference

A class to represent a single logging mechanism to a logging service on the network.

```
#include <be_io_syslogsheet.h>
```

Inheritance diagram for BiometricEvaluation::IO::SysLogsheet:



Public Member Functions

- SysLogsheet (const std::string &url, const std::string &description, const std::string &appname, bool sequenced, bool utc)

  Create a new log sheet.
- SysLogsheet (const std::string &url, const std::string &description, const std::string &appname, const std::string &hostname, bool sequenced, bool utc)

  Create a new log sheet.
- ~SysLogsheet ()
- void write (const std::string &entry)

  Write a string as an entry to the backing store.
- void writeComment (const std::string &entry)

  Write a string as a comment to the backing store.
- void writeDebug (const std::string &entry)

  Write a string as a debug entry to the backing store.
- void sync ()

  Synchronize any buffered data to the underlying backing store.

Protected Member Functions

- SysLogsheet (const SysLogsheet &)
- SysLogsheet & operator= (const SysLogsheet &)
- void setup (const std::string &url, const std::string &description)
- void writeToLogger (const std::string &priority, const char delimiter, const std::string &prefix, const std::string &message)

Protected Attributes

- std::string **_hostname**
- std::string **_appname**
- std::string **_procid**
- int _sockFD
- bool _sequenced
- bool _operational
- bool _utc

Additional Inherited Members

H.111.1 Detailed Description

A class to represent a single logging mechanism to a logging service on the network.

Log entries are sent to the logging server in RFC5424 format with a timestamp of the local system in UTC. Normal and comment entries are sent to the logger with a PRI field indicating the 'local0' facility and a severity of 'Informational'. Debug entries are sent with facility of 'local1' and severity 'Debug'. A basic syslog config file would contain these lines: local0.info /var/log/info.log local1.debug /var/log/debug.log

The hostname is added to each entry but may be overridden by constructing the object with a given hostname, including the RFC5424 NILVALUE character. The PROCID part of each log message will be filled in with the process ID. Multi-line messages are segmented and sent the to logger as separate entries with the same timestamp and sequence number.

H.111.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::SysLogsheet::SysLogsheet (const std::string & *url*, const std::string & *description*, const std::string & *appname*, bool *sequenced*, bool *utc*)

Create a new log sheet.

Parameters

in	*url*	The Uniform Resource Locator describing the logging service. Accepted forms are syslog://hostname:port
in	*description*	The text used to describe the sheet. This text is written into the log prior to any entries.
in	*appname*	The name of the application. This text is written into each log entry.
in	*sequenced*	True if each entry should include a sequence number, false if not.
in	*utc*	True if timestamps should be in Coordinated Universal Time (UTC), false for local time.

Exceptions

Error::StrategyError	An error occurred when connecting to the logging system, or URL is malformed.

BiometricEvaluation::IO::SysLogsheet::SysLogsheet (const std::string & *url*, const std::string & *description*, const std::string & *appname*, const std::string & *hostname*, bool *sequenced*, bool *utc*)

Create a new log sheet.

Parameters

in	*url*	The Uniform Resource Locator describing the logging service. Accepted forms are syslog://hostname:port
in	*description*	The text used to describe the sheet. This text is written into the log prior to any entries.
in	*appname*	The name of the application. This text is written into each log entry.
in	*hostname*	The string to use as the hostname for all log entries.
in	*sequenced*	True if each entry should include a sequence number, false if not.

in	*utc*	True if timestamps should be in Coordinated Universal Time (UTC), false for local time.

Exceptions

Error::StrategyError	An error occurred when connecting to the logging system, or URL is malformed.

BiometricEvaluation::IO::SysLogsheet::~SysLogsheet ( )

Destructor

BiometricEvaluation::IO::SysLogsheet::SysLogsheet (const SysLogsheet &) [protected]

Prevent copying of SysLogsheet objects

H.111.3 Member Function Documentation

SysLogsheet& BiometricEvaluation::IO::SysLogsheet::operator= (const SysLogsheet &) [protected]

Prevent copying of SysLogsheet objects

void BiometricEvaluation::IO::SysLogsheet::setup (const std::string & *url*, const std::string & *description*) [protected]

Helper function to build connections

void BiometricEvaluation::IO::SysLogsheet::sync ( ) [virtual]

Synchronize any buffered data to the underlying backing store.

This syncing is dependent on the behavior of the underlying storage mechanism.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

Reimplemented from BiometricEvaluation::IO::Logsheet.

void BiometricEvaluation::IO::SysLogsheet::write (const std::string & *entry*) [virtual]

Write a string as an entry to the backing store.

This does not affect the current log entry buffer, but does increment the entry number.

Parameters

in	*entry*	The text of the log entry.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

Reimplemented from BiometricEvaluation::IO::Logsheet.

void BiometricEvaluation::IO::SysLogsheet::writeComment (const std::string & *entry*) [virtual]

Write a string as a comment to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

in	*entry*	The text of the comment.

Exceptions

Error::StrategyError	An error occurred when using the underlying backing store.

Reimplemented from BiometricEvaluation::IO::Logsheet.

void BiometricEvaluation::IO::SysLogsheet::writeDebug (const std::string & *entry*) [virtual]

Write a string as a debug entry to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A debug line is prefixed with DebugDelimiter followed by a space.

Parameters

in	*entry*	The text of the debug message.

Exceptions

Error::StrategyError	An error occurred when logging.

Reimplemented from BiometricEvaluation::IO::Logsheet.

void BiometricEvaluation::IO::SysLogsheet::writeToLogger (const std::string & *priority*, const char *delimiter*, const std::string & *prefix*, const std::string & *message*) [protected]

Helper function to write to the logger

H.111.4 Member Data Documentation

bool BiometricEvaluation::IO::SysLogsheet::_operational [protected]

Whether the sheet is operational

bool BiometricEvaluation::IO::SysLogsheet::_sequenced [protected]

Whether to include entry sequence numbers

int BiometricEvaluation::IO::SysLogsheet::_sockFD [protected]

Socket file descriptor for the logging system

bool BiometricEvaluation::IO::SysLogsheet::_utc [protected]

Whether time stamps are in UTC

H.112 BiometricEvaluation::MPI::TaskCommand Class Reference

The command given to an MPI task.

```
#include <be_mpi.h>
```

Public Types

- enum Kind {
  Continue = 0, Ignore = 1, Exit = 2, QuickExit = 3,
  TermExit = 4 }

H.112.1 Detailed Description

The command given to an MPI task.

H.112.2 Member Enumeration Documentation

enum BiometricEvaluation::MPI::TaskCommand::Kind

Enumerator

Ignore Normal operation.

Exit Ignore the message.

QuickExit Transition to the normal shutdown state.

TermExit Transition to the quick shutdown state.

H.113 BiometricEvaluation::MPI::TaskStatus Class Reference

The status of an MPI distributor or receiver task.

`#include <be_mpi.h>`

Public Types

- enum Kind { **OK** = 0, Failed = 1, Exit = 2 }

H.113.1 Detailed Description

The status of an MPI distributor or receiver task.

H.113.2 Member Enumeration Documentation

enum BiometricEvaluation::MPI::TaskStatus::Kind

Enumerator

Failed Normal operation.

Exit Failed to complete an operation.

H.114 BiometricEvaluation::Time::Timer Class Reference

This class can be used by applications to report the amount of time a block of code takes to execute.

`#include <be_time_timer.h>`

Public Types

- using BE_CLOCK_TYPE = std::chrono::steady_clock

Public Member Functions

- Timer ()
- void start ()

  Start tracking time.
- void stop ()

  Stop tracking time.
- uint64_t elapsed () const

  Get the elapsed time in microseconds between calls to this object's start() and stop() methods.
- std::string elapsedStr (bool displayUnits=false) const

  Convenience method for printing elapsed time as a string.

H.114.1 Detailed Description

This class can be used by applications to report the amount of time a block of code takes to execute.

Applcations wrap the block of code in the Timer::start() and Timer::stop() calls, then use Timer::elapsed() to obtain the calculated time of the operation.

Warning

Timers are not threadsafe and should only be used to time operations within the same thread.

H.114.2 Member Typedef Documentation

using BiometricEvaluation::Time::Timer::BE_CLOCK_TYPE = std::chrono::steady_clock

Clock type to use, aliased for easy replacement.

H.114.3 Constructor & Destructor Documentation

BiometricEvaluation::Time::Timer::Timer ( )

Constructor for the Timer object.

H.114.4 Member Function Documentation

uint64_t BiometricEvaluation::Time::Timer::elapsed ( ) const

Get the elapsed time in microseconds between calls to this object's start() and stop() methods.

Returns

The number of microseconds between calls to this object's start() and stop() methods.

Exceptions

Error::StrategyError	This object is currently timing an operation or an error occurred when obtaining timing information.

std::string BiometricEvaluation::Time::Timer::elapsedStr (bool *displayUnits* = `false`) const

Convenience method for printing elapsed time as a string.

Parameters

displayUnits	Append the elapsed time units.

Returns

String representing the elapsed time.

Exceptions

Error::StrategyError	Propagated from elapsed().

void BiometricEvaluation::Time::Timer::start ( )

Start tracking time.

Exceptions

Error::StrategyError	This object is currently timing an operation or an error occurred when obtaining timing information.

void BiometricEvaluation::Time::Timer::stop ( )

Stop tracking time.

Exceptions

Error::StrategyError	This object is not currently timing an operation or an error occurred when obtaining timing information.

H.115 BiometricEvaluation::View::View Class Reference

A class to represent single biometric element view.

`#include <be_view_view.h>`

Inheritance diagram for BiometricEvaluation::View::View:

Public Member Functions

- std::shared_ptr< Image::Image > getImage () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- Image::Size getImageSize () const

  Obtain the image size.
- Image::Resolution getImageResolution () const

  Obtain the image resolution.
- uint32_t getImageDepth () const

  Obtain the image depth.
- Image::CompressionAlgorithm getCompressionAlgorithm () const

  Obtain the compression algorithm used on the image.

- Image::Resolution getScanResolution () const

  Obtain the image scan resolution.

Protected Member Functions

- void setImageSize (const BiometricEvaluation::Image::Size &imageSize)

  Mutator for the image size.
- void setImageDepth (uint32_t imageDepth)

  Mutator for the image size.
- void setImageResolution (const BiometricEvaluation::Image::Resolution &imageResolution)

  Mutator for the image resolution.
- void setScanResolution (const BiometricEvaluation::Image::Resolution &scanResolution)

  Mutator for the image scan resolution.
- void setImageData (const BiometricEvaluation::Memory::uint8Array &imageData)

  Mutator for the image data.
- void setCompressionAlgorithm (const Image::CompressionAlgorithm &ca)

  Mutator for the compression algorithm.

H.115.1 Detailed Description

A class to represent single biometric element view.

Included in a view is the biometric image and any derived information, such as minutiae points.

H.115.2 Member Function Documentation

Image::CompressionAlgorithm BiometricEvaluation::View::View::getCompressionAlgorithm ( ) const

Obtain the compression algorithm used on the image.

This value is as present in the biometric record, and not obtained from the image data itself.

Returns

The compression algorithm.

std::shared_ptr<Image::Image> BiometricEvaluation::View::View::getImage ( ) const

Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)

Not all views will have an image, however the derived information, such as minutiae, may be present.

Returns

The image data.

uint32_t BiometricEvaluation::View::View::getImageDepth ( ) const

Obtain the image depth.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image depth must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

Returns

The image depth.

Image::Resolution BiometricEvaluation::View::View::getImageResolution ( ) const

Obtain the image resolution.

Image resolution is taken from the biometric record, and not from the image data.

Returns

The scan resolution.

Note

In some cases, the resolution may be the components of the pixel ratio, and applications must check the Image::Resolution::Units field for value NA.

Image::Size BiometricEvaluation::View::View::getImageSize ( ) const

Obtain the image size.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image size must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

Returns

The image size.

Image::Resolution BiometricEvaluation::View::View::getScanResolution ( ) const

Obtain the image scan resolution.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image resolution must be equal, but applications can check for inconsistencies.

Returns

The scan resolution.

Note

In some cases, the resolution may be the components of the pixel ratio, and applications must check the Image::Resolution::Units field for value NA.

void BiometricEvaluation::View::View::setImageData (const BiometricEvaluation::Memory::uint8↩ Array & *imageData*) [protected]

Mutator for the image data.

Parameters

in	*imageData*	The image data object.

void BiometricEvaluation::View::View::setImageDepth (uint32_t *imageDepth*) [protected]

Mutator for the image size.
Parameters

in	*imageDepth*	The image depth.

void BiometricEvaluation::View::View::setImageResolution (const BiometricEvaluation::Image::↩ Resolution & *imageResolution*) [protected]

Mutator for the image resolution.
Parameters

in	*image*↩ *Resolution*	The image resolution object.

void BiometricEvaluation::View::View::setImageSize (const BiometricEvaluation::Image::Size & *imageSize*) [protected]

Mutator for the image size.
Parameters

in	*imageSize*	The image size object.

void BiometricEvaluation::View::View::setScanResolution (const BiometricEvaluation::Image::↩ Resolution & *scanResolution*) [protected]

Mutator for the image scan resolution.
Parameters

in	*scanResolution*	The image scan resolution object.

H.116 BiometricEvaluation::Time::Watchdog Class Reference

A Watchdog object can be used by applications to limit the amount of processing time taken by a block of code.

```
#include <be_time_watchdog.h>
```

Public Member Functions

- Watchdog (const uint8_t type)
- void setInterval (uint64_t interval)
- void start ()
- void stop ()
- bool expired ()
- void setCanSigJump ()
- void clearCanSigJump ()

- void setExpired ()
- void clearExpired ()

Static Public Attributes

- static const uint8_t PROCESSTIME = 0
- static const uint8_t REALTIME = 1
- static bool **_canSigJump**
- static sigjmp_buf **_sigJumpBuf**

H.116.1 Detailed Description

A Watchdog object can be used by applications to limit the amount of processing time taken by a block of code.

A Watchdog object is used to set a timer that, upon expiration, will force a jump to a location within the process. An application can detect whether the timer expired at that point in the code. Watchdog builds on the POSIX setitimer(2) call. Timer intervals are in terms of process virtual time or real time, based on how the object is constructed.

Most applications will not directly invoke the methods of the WatchDog class, instead using the BEGIN_WATCHDOG_BLOCK() and END_WATCHDOG_BLOCK() macros. Applications should not install their own signal handlers, but use the SignalManager class instead.

The BEGIN_WATCHDOG_BLOCK() macro sets up the jump block and tells the Watchdog object to start handling the alarm signal. Applications must call setInterval() before invoking the BEGIN_WATCHDOG_BLOCK() macro.

The END_WATCHDOG_BLOCK() macro disables the watchdog timer, but doesn't affect the assigned interval value. Applications can set the interval once and use the block macros repeatedly. Failure to call setInterval() results in an effectively disabled timer, as does setting the interval to 0.

The ABORT_WATCHDOG() macro also disables the watchdog timer but does not create the code point destination for the jump point. This macro should be used to disable a Watchdog object when the application is no longer interested in the timeout condition.

Attention

> The BEGIN_WATCHDOG_BLOCK() macro must be paired with either the END_WATCHDOG_BLOCK() macro or ABORT_WATCHDOG_BLOCK() macro. Failure to do so may result in undefined behavior as a running Watchdog timer may expire, forcing a jump into an incompletely initialized function.

Note

> Process virtual timing may not be available on all systems. In those cases, an application compilation error will occur because PROCESSTIME will not be defined.

Attention

> On many systems, the sleep(3) call is implemented using alarm signals, the same technique used by the Watchdog class. Therefore, applications should not call sleep(3) inside the Watchdog block; behavior is undefined in that case, but usually results in cancellation of the Watchdog timer.
> The setCanSigJump(), clearCanSigJump(), setExpired() and clearExpired() methods are not meant to be used directly by applications, which should use the BEGIN_WATCHDOG_BLOCK()/END_WATCHDOG_BLOCK() macro pair.

See also

> Error::SignalManager

H.116.2 Constructor & Destructor Documentation

BiometricEvaluation::Time::Watchdog::Watchdog (const uint8_t *type*)

Construct a new Watchdog object.

Parameters

in	*type*	The type of timer, ProcessTime or RealTime.

Exceptions

Error::NotImplemented	The type of watchdog requested is not implemented.
Error::ParameterError	The type is invalid.

Warning

Watchdog::PROCESSTIME is not supported under Cygwin.

H.116.3 Member Function Documentation

void BiometricEvaluation::Time::Watchdog::clearCanSigJump ( )

Clears the flag for the Watchdog object to indicate that the signal jump block is no longer valid.

void BiometricEvaluation::Time::Watchdog::clearExpired ( )

Clear the flag indicating the timer expired.

bool BiometricEvaluation::Time::Watchdog::expired ( )

Indicate whether the watchdog timer expired.

Returns

true if the timer expired, false otherwise.

void BiometricEvaluation::Time::Watchdog::setCanSigJump ( )

Indicate that the signal handler can jump into the application code after handling the signal.

void BiometricEvaluation::Time::Watchdog::setExpired ( )

Set a flag to indicate the timer expired.

void BiometricEvaluation::Time::Watchdog::setInterval (uint64_t *interval*)

Set the interval for the timer, but don't start the timer. Setting a value of 0 will essentially disable the timer. Timer intervals are in microseconds, however actual intervals are dependent on the resolution of the system clock, and may not be at microsecond resolution.

Parameters

in	*interval*	The timer interval, in microseconds.

void BiometricEvaluation::Time::Watchdog::start ( )

Start a watchdog timer.

Exceptions

Error::StrategyError	Could not register the signal handler, or could not create the timer.

void BiometricEvaluation::Time::Watchdog::stop ( )

Stop a watchdog timer.
Exceptions

Error::StrategyError	Could not clear the timer.

H.116.4 Member Data Documentation

const uint8_t BiometricEvaluation::Time::Watchdog::PROCESSTIME = 0 [static]

A Watchdog based on process time.

const uint8_t BiometricEvaluation::Time::Watchdog::REALTIME = 1 [static]

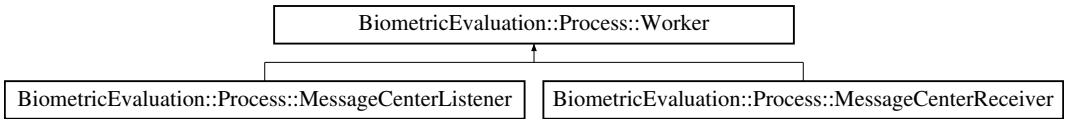
A Watchdog based on real (wall clock) time.

H.117 BiometricEvaluation::Process::Worker Class Reference

An abstraction of an instance that performs work on given data.

`#include <be_process_worker.h>`

Inheritance diagram for BiometricEvaluation::Process::Worker:



Public Member Functions

- virtual int32_t workerMain ()=0

  The method that will get called to start execution by a ProcessManager.
- std::shared_ptr< void > getParameter (const std::string &name)

  Obtain a parameter passed to this Worker.
- double getParameterAsDouble (const std::string &name)

  Obtain a parameter passed to this Worker as a double.
- int64_t getParameterAsInteger (const std::string &name)

  Obtain a parameter passed to this Worker as an integer.
- std::string getParameterAsString (const std::string &name)

  Obtain a parameter passed to this Worker as a string.
- void setParameter (const std::string &name, std::shared_ptr< void > argument)

  Pass a parameter to this Worker.
- void stop ()

  Tell this Worker to return ASAP.
- void closeWorkerPipeEnds ()

Perform initialization for communication from Worker to Manager.

- void closeManagerPipeEnds ()

  Perform initialization for communication from Manager to Worker.
- int getSendingPipe () const

  Obtain the pipe used to send messages to this Worker.
- int getReceivingPipe () const

  Obtain the pipe used to receive messages to this Worker.
- void sendMessageToManager (const Memory::uint8Array &message)

  Send a message to the Manager.
- void receiveMessageFromManager (Memory::uint8Array &message)

  Receive a message from the Manager.
- void _initCommunication ()

  Perform general communication initialization from Constructor.
- virtual ~Worker ()

  Worker destructor.

Protected Member Functions

- Worker ()

  Worker constructor.
- bool stopRequested () const

  Determine if the parent has requested this child to exit.
- bool waitForMessage (int numSeconds=-1) const

  Block while waiting for a message from the Manager.

H.117.1 Detailed Description

An abstraction of an instance that performs work on given data.

H.117.2 Member Function Documentation

void BiometricEvaluation::Process::Worker::_initCommunication ( )

Perform general communication initialization from Constructor.

Exceptions

Error::StrategyError	Error in initialization.

void BiometricEvaluation::Process::Worker::closeManagerPipeEnds ( )

Perform initialization for communication from Manager to Worker.

Note

Behavior is undefined if called by a non-Worker.

Exceptions

Error::StrategyError	Communications not enabled.

void BiometricEvaluation::Process::Worker::closeWorkerPipeEnds ( )

Perform initialization for communication from Worker to Manager.

Note

Behavior is undefined if called by a non-Manager.

Exceptions

Error::StrategyError	Communications not enabled.

std::shared_ptr<void> BiometricEvaluation::Process::Worker::getParameter (const std::string & *name*)

Obtain a parameter passed to this Worker.

Parameters

name	The parameter name to retrieve.

Returns

shared_ptr to the parameter argument.

Exceptions

std::out_of_range	name was not set.

double BiometricEvaluation::Process::Worker::getParameterAsDouble (const std::string & *name*)

Obtain a parameter passed to this Worker as a double.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as a double.

Exceptions

std::out_of_range	name was not set.

int64_t BiometricEvaluation::Process::Worker::getParameterAsInteger (const std::string & *name*)

Obtain a parameter passed to this Worker as an integer.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as an integer.

Exceptions

std::out_of_range	name was not set.

std::string BiometricEvaluation::Process::Worker::getParameterAsString (const std::string & *name*)

Obtain a parameter passed to this Worker as a string.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as a string.

Exceptions

std::out_of_range	name was not set.

int BiometricEvaluation::Process::Worker::getReceivingPipe ( ) const

Obtain the pipe used to receive messages to this Worker.

Returns

Receiving pipe.

Exceptions

Error::ObjectDoesNotExist	Worker exiting soon, communication disabled.
Error::StrategyError	Communications not enabled.

int BiometricEvaluation::Process::Worker::getSendingPipe ( ) const

Obtain the pipe used to send messages to this Worker.

Returns

Sending pipe.

Exceptions

Error::ObjectDoesNot↩ Exist	Worker exiting soon, communication disabled.
Error::StrategyError	Communications not enabled.

void BiometricEvaluation::Process::Worker::receiveMessageFromManager (Memory::uint8Array & *message*)

Receive a message from the Manager.

Parameters

out	*message*	Buffer to store the received message.

Exceptions

Error::ObjectDoesNot↩ Exist	Widowed pipe.
Error::StrategyError	Communications not enabled.

See also

waitForMessage

void BiometricEvaluation::Process::Worker::sendMessageToManager (const Memory::uint8Array & *message*)

Send a message to the Manager.

Parameters

in	*message*	Message to send.

Exceptions

Error::ObjectDoesNot↩ Exist	Widowed pipe.
Error::StrategyError	Communications not enabled.

void BiometricEvaluation::Process::Worker::setParameter (const std::string & *name*, std::shared_ptr< void > *argument*)

Pass a parameter to this Worker.

Parameters

name	A unique identifier for this parameter
argument	A shared_ptr to the object to store.

void BiometricEvaluation::Process::Worker::stop ( )

Tell this Worker to return ASAP.

Attention

This method should not be overridden.

bool BiometricEvaluation::Process::Worker::stopRequested ( ) const [protected]

Determine if the parent has requested this child to exit.

Returns

Whether or not this child should exit.

Attention

This method should not be overridden.

bool BiometricEvaluation::Process::Worker::waitForMessage (int *numSeconds* = *-1*) const [protected]

Block while waiting for a message from the Manager.

Parameters

numSeconds	Number of seconds to wait for a message, or any value < 0 to wait forever.

Returns

true once a message is ready to be read or false if an error occured.

virtual int32_t BiometricEvaluation::Process::Worker::workerMain ( ) [pure virtual]

The method that will get called to start execution by a ProcessManager.

Returns

Status code.

Note

If an object of this class is added to a Process::ForkManager object, the implementation of Process::Worker::workerMain() should release all resources prior to returning.

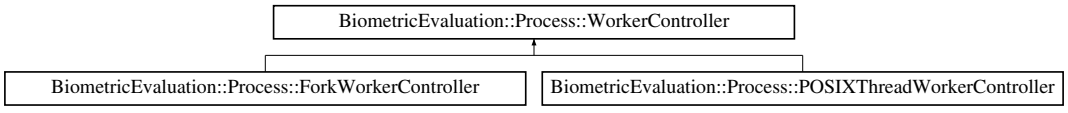
Implemented in BiometricEvaluation::Process::MessageCenterReceiver, and BiometricEvaluation::Process::MessageCenterListener.

H.118 BiometricEvaluation::Process::WorkerController Class Reference

Wrapper of a Worker returned from a Process::Manager.

`#include <be_process_workercontroller.h>`

Inheritance diagram for BiometricEvaluation::Process::WorkerController:



Public Member Functions

- WorkerController (std::shared_ptr< Worker > worker)
- virtual void sendMessageToWorker (const Memory::uint8Array &message)

  Send a message to the Worker contained within this WorkerController.
- virtual void setParameter (const std::string &name, std::shared_ptr< void > argument)

  Set the parameter to be passed to the Worker.
- virtual void setParameterFromDouble (const std::string &name, double argument)

  Set a double parameter to be passed to the Worker.
- virtual void setParameterFromInteger (const std::string &name, int64_t argument)

  Set an integer parameter to be passed to the Worker.
- virtual void setParameterFromString (const std::string &name, const std::string &argument)

  Set a string parameter to be passed to the Worker.
- virtual void reset ()

  Reuse the Worker.
- virtual bool isWorking () const =0

  Obtain whether or not Worker is working.
- virtual bool everWorked () const =0

  Obtain whether or not this Worker has ever worked.
- bool finishedWorking () const

  Obtain whether or not this Worker has both started and finished its task.
- std::shared_ptr< Worker > getWorker () const

  Obtain the Worker instance being wrapped.
- virtual ~WorkerController ()

  WorkerController destructor.

Protected Attributes

- std::shared_ptr< Worker > _worker

H.118.1 Detailed Description

Wrapper of a Worker returned from a Process::Manager.

H.118.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::WorkerController::WorkerController (std::shared_ptr< Worker > *worker*)

WorkerController constructor.
Parameters

worker	The Worker instance to wrap.

H.118.3 Member Function Documentation

virtual bool BiometricEvaluation::Process::WorkerController::everWorked ( ) const [pure virtual]

Obtain whether or not this Worker has ever worked.

Returns

true the Worker has ever or is currently working, false otherwise.

Note

reset() will change the result of this method.

Implemented in BiometricEvaluation::Process::ForkWorkerController, and BiometricEvaluation::Process::POSIXThreadWorkerController.

bool BiometricEvaluation::Process::WorkerController::finishedWorking ( ) const [inline]

Obtain whether or not this Worker has both started and finished its task.

Returns

true if the Worker has both started and finished performing its task, false otherwise.

Note

reset() will change the result of this method.

std::shared_ptr<Worker> BiometricEvaluation::Process::WorkerController::getWorker ( ) const

Obtain the Worker instance being wrapped.

Returns

Worker instance.

virtual bool BiometricEvaluation::Process::WorkerController::isWorking ( ) const [pure virtual]

Obtain whether or not Worker is working.

Returns

Whether or not the Worker is working.

Implemented in BiometricEvaluation::Process::ForkWorkerController, and BiometricEvaluation::Process::POSIXThreadWorkerController.

virtual void BiometricEvaluation::Process::WorkerController::reset ( ) [virtual]

Reuse the Worker.

Exceptions

Error::ObjectExists	The previously started Worker is still running.

Reimplemented in BiometricEvaluation::Process::ForkWorkerController, and BiometricEvaluation::Process::POSIXThreadWorkerController.

virtual void BiometricEvaluation::Process::WorkerController::sendMessageToWorker (const Memory::uint8Array & *message*) [virtual]

Send a message to the Worker contained within this WorkerController.
Parameters

message	Message to send to the Worker.

Exceptions

Error::ObjectDoesNotExist	Worker receive pipe is closed (Worker object likely destroyed).
Error::StrategyError	Message sending failed.

virtual void BiometricEvaluation::Process::WorkerController::setParameter (const std::string & *name*, std::shared_ptr< void > *argument*) [virtual]

Set the parameter to be passed to the Worker.
Parameters

in	*name*	The name representing the argument in the Worker.
in	*argument*	The argument to be passed to the Worker.

Note

Subsequent calls to setParameter() with the same name will overwrite any exiting argument.

virtual void BiometricEvaluation::Process::WorkerController::setParameterFromDouble (const std::string & *name*, double *argument*) [virtual]

Set a double parameter to be passed to the Worker.
Parameters

in	*name*	The name representing the argument in the Worker.
in	*argument*	The double to be passed to the Worker.

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

virtual void BiometricEvaluation::Process::WorkerController::setParameterFromInteger (const std::string & *name*, int64_t *argument*) [virtual]

Set an integer parameter to be passed to the Worker.

Parameters

in	*name*	The name representing the argument in the Worker.
in	*argument*	The integer to be passed to the Worker.

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

virtual void BiometricEvaluation::Process::WorkerController::setParameterFromString (const std::string & *name*, const std::string & *argument*) [virtual]

Set a string parameter to be passed to the Worker.
Parameters

in	*name*	The name representing the argument in the Worker.
in	*argument*	The string to be passed to the Worker.

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

H.118.4 Member Data Documentation

std::shared_ptr<Worker> BiometricEvaluation::Process::WorkerController::_worker [protected]

The Worker instance that is running in this child

H.119 BiometricEvaluation::MPI::WorkPackage Class Reference

A class to represent a piece of work to be acted upon by a processor.
`#include <be_mpi_workpackage.h>`

Public Member Functions

- WorkPackage ()
  Construct an empty work package.
- WorkPackage (const Memory::uint8Array &data)
  Construct a work package with some data.
- void getData (Memory::uint8Array &data) const
  Obtain the package data in raw form.
- void setData (const Memory::uint8Array &data)
  Set the package data from raw data.
- uint64_t getSize () const
  Obtain the size of the package data.
- uint64_t getNumElements () const
  Obtain the number of elements in the package.
- void setNumElements (const uint64_t numElements)
  Set the number of elements in the package.

H.119.1 Detailed Description

A class to represent a piece of work to be acted upon by a processor.

The work package is an wrapper around the data to be processed, along with some ancillary information.

H.119.2 Constructor & Destructor Documentation

BiometricEvaluation::MPI::WorkPackage::WorkPackage (const Memory::uint8Array & *data*)

Construct a work package with some data.

Parameters

in	*data*	The data that will be managed by this work package.

H.119.3 Member Function Documentation

uint64_t BiometricEvaluation::MPI::WorkPackage::getNumElements ( ) const

Obtain the number of elements in the package.

This value is determined by the application and must be set therein, otherwise 0 is returned.

Returns

The number of application defined elements in the work package.

uint64_t BiometricEvaluation::MPI::WorkPackage::getSize ( ) const

Obtain the size of the package data.

Returns

The size (in octets) of the raw data item.

void BiometricEvaluation::MPI::WorkPackage::setData (const Memory::uint8Array & *data*)

Set the package data from raw data.

Parameters

in	*data*	The data copied into the work package.

void BiometricEvaluation::MPI::WorkPackage::setNumElements (const uint64_t *numElements*)

Set the number of elements in the package.

Parameters

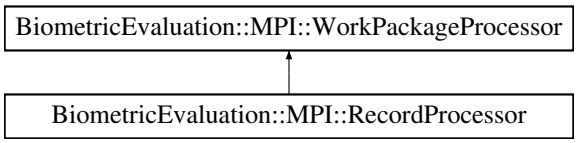
in	*numElements*	The number of appplication-defined elements in the work package.

H.120 BiometricEvaluation::MPI::WorkPackageProcessor Class Reference

Represents an object that processes the contents of a work package.

`#include <be_mpi_workpackageprocessor.h>`

Inheritance diagram for BiometricEvaluation::MPI::WorkPackageProcessor:



Public Member Functions

- virtual std::shared_ptr < WorkPackageProcessor > newProcessor (std::shared_ptr< IO::Logsheet > &logsheet)=0

  Obtain an object that will process work packages. This method is part of the factory personality.
- virtual void performInitialization (std::shared_ptr< IO::Logsheet > &logsheet)=0

  Initialization function to be called before work is distributed to the work package processor.
- virtual void processWorkPackage (MPI::WorkPackage &workPackage)=0

  Process the data contents of the work package. This method is part of the worker personality.
- void setLogsheet (std::shared_ptr< IO::Logsheet > &logsheet)

  Set the IO::Logsheet object that can be used to save message for objects of this class.
- std::shared_ptr< IO::Logsheet > getLogsheet ()

  Obtain the IO::Logsheet object that can be used to save message for objects of this class.

H.120.1 Detailed Description

Represents an object that processes the contents of a work package.

A WorkPackageProcessor presents two personalities: One that of a worker to process work packages, and one that is a factory to return worker objects of the implementation class.

Subclasses of this class implement the functionality needed to perform an action on the work package data. The processing done by the implementation is application and data type specific.

Ultimately, the final implementation of the WorkPackageProcessor class is done in the application. Access to the Logsheet object maintained by the framework is provided by this class.

H.120.2 Member Function Documentation

std::shared_ptr<IO::Logsheet> BiometricEvaluation::MPI::WorkPackageProcessor::getLogsheet ( )

Obtain the IO::Logsheet object that can be used to save message for objects of this class.

Returns

logsheet A shared pointer to the Logsheet object.

virtual std::shared_ptr<WorkPackageProcessor> BiometricEvaluation::MPI::Work↩ PackageProcessor::newProcessor (std::shared_ptr< IO::Logsheet > & *logsheet*) [pure virtual]

Obtain an object that will process work packages. This method is part of the factory personality.

Parameters

logsheet	A shared pointer to the IO::Logsheet that may be used to save messages generated by the object.

Returns

A shared pointer to the work package processor.

Implemented in BiometricEvaluation::MPI::RecordProcessor.

virtual void BiometricEvaluation::MPI::WorkPackageProcessor::performInitialization (std::shared_ptr< IO::Logsheet > & *logsheet*) [pure virtual]

Initialization function to be called before work is distributed to the work package processor.

Implementations of this class can use this function to do any processing necessary before work is given to the processor, pre-forking.

This method is part of the factory personality. All state that is to be common across all package processor objects can be initialized in this method.

Parameters

logsheet	A shared pointer to the IO::Logsheet that may be used to save messages generated by the object.

Exceptions

Error::Exception	An implementation specific. error occurred.

Implemented in BiometricEvaluation::MPI::RecordProcessor.

virtual void BiometricEvaluation::MPI::WorkPackageProcessor::processWorkPackage (MPI::WorkPackage & *workPackage*) [pure virtual]

Process the data contents of the work package. This method is part of the worker personality.

Parameters

in	*workPackage*	The work package.

Exceptions

Error::Exception	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.

Implemented in BiometricEvaluation::MPI::RecordProcessor.

void BiometricEvaluation::MPI::WorkPackageProcessor::setLogsheet (std::shared_ptr< IO::Logsheet > & *logsheet*)

Set the IO::Logsheet object that can be used to save message for objects of this class.

Parameters

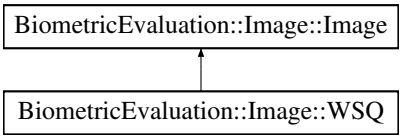
in	*logsheet*	A shared pointer to the Logsheet object.

H.121 BiometricEvaluation::Image::WSQ Class Reference

A WSQ-encoded image.

`#include <be_image_wsq.h>`

Inheritance diagram for BiometricEvaluation::Image::WSQ:



Public Member Functions

- **WSQ** (const uint8_t *data, const uint64_t size)
- Memory::uint8Array getRawData () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- Memory::uint8Array getRawGrayscaleData (uint8_t depth=8) const

  Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isWSQ (const uint8_t *data, uint64_t size)

Additional Inherited Members

H.121.1 Detailed Description

A WSQ-encoded image.

H.121.2 Member Function Documentation

Memory::uint8Array BiometricEvaluation::Image::WSQ::getRawData ( ) const `[virtual]`

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError	Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

Memory::uint8Array BiometricEvaluation::Image::WSQ::getRawGrayscaleData (uint8_t *depth* = `8`) const `[virtual]`

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 8 or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError	Error decompressing image data.
Error::ParameterError	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

static bool BiometricEvaluation::Image::WSQ::isWSQ (const uint8_t * *data*, uint64_t *size*) [static]

Whether or not data is a WSQ image.
Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a WSQ image, false otherwise

Index