Biometric Evaluation Common Framework

Wayne Salamon and Greg Fiumara

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 [11].

1.1 Rationale

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

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

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

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

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

Chapter 2

Overview

The Biometric Evaluation Framework (BECommon) is a set of C++[13] classes, error codes, and design patterns used to create a common environment to provide logging, data management, error handling, and other functionality that is needed for many applications used in the testing of biometric software. The goals of the framework include:

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

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

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* name space.

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

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

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

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

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

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

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

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

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

BECommon is designed to reduce the intra-framework dependencies.

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

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

Chapter 3

Framework

The Framework package is used to retrieve information about the Biometric Evaluation Framework itself. Framework version, compiler, and other information can be queried by applications, mostly for logging so runtime information can be captured. The intent at this time is *not* to control the linkage of applications against specific versions of the framework.

Listing 3.1: Using a the Framework API

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

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 [10]. In these cases, functions exist to allocate and free blocks of memory, and these calls must be made by the applications which use those libraries. To assist BECommon clients that use these existing libraries, the AutoBuffer class wraps the C memory management functions, guaranteeing the release of C objects when the AutoBuffer goes out of scope.

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

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

Listing 4.1: Using the AutoBuffer

```
1  #include <be_memory_autobuffer.h>
2  #include <iostream>
3  extern "C" {
4    #include <an2k.h>
5  }
6
7  int
8  main(int argc, char* argv[]) {
9
```

```
10
11      /*
12       * alloc_ANSI_NIST(), free_ANSI_NIST(), and copy_ANSI_NIST()
13       * are functions in the NBIS AN2K library.
14       */
15      Memory::AutoBuffer<ANSI_NIST> an2k =
16          Memory::AutoBuffer<ANSI_NIST>(&alloc_ANSI_NIST,
17              &free_ANSI_NIST, &copy_ANSI_NIST);
18      if (read_ANSI_NIST(fp, an2k) != 0) {
19              cerr << "Could not read AN2K file." << endl;
20              return (EXIT_FAILURE);
21      }
22
23      for (int i = 1; i < an2k->num_records; i++) {
24          // process the ANSI/NIST record ...
25      }
26  }
```

4.2 AutoArray

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

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

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

Listing 4.2: Using AutoArray with RecordStores

```
1  #include <be_io_dbrecstore.h>
2  #include <be_memory_autoarray.h>
3
4  #include <iostream>
5
6  using namespace BiometricEvaluation;
7
8  int
9  main(
10     int argc,
11     char *argv[])
12 {
```

```
13          IO::DBRecordStore rs("db_recstore", ".", IO::READONLY);
14
15          uint64_t value_size = 0;
16          string key("");
17          Memory::AutoArray<uint8_t> value;
18          for (bool stop = false; stop == false; ) {
19                  try {
20                          // Non-destructively resize the AutoArray to
                                hold
21                          // the next record.
22                          value.resize(rs.sequence(key, NULL));
23
24                          // Read the record into the AutoArray (treats
                                the
25                          // AutoArray as a pointer).
26                          rs.read(key, value);
27
28                          // Do something with value.
29                          std::cout << "Key " << key << " has a value of
                                " <<
30                              value.size() << " bytes" << std::endl;
31                  } catch (Error::ObjectDoesNotExist) {
32                          stop = true;
33                  }
34          }
35
36          return (0);
37  }
```

AutoArray is adapted from "c_array" [13, 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 19 shows an example of exception handling when using the logging classes described in Section 6.3 on page 19.

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

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

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

```
1  #include <be_error_signal_manager.h>
2  using namespace BiometricEvaluation;
3
4  int main(int argc, char *argv[])
5  {
6      Error::SignalManager *sigmgr = new Error::SignalManager();
7
8      sigset_t sigset;
9      sigemptyset(&sigset);
10     sigaddset(&sigset, SIGUSR1);
11     sigmgr->setSignalSet(sigset);
12
13     FILE *fp = fopen( ... );
14     BEGIN_SIGNAL_BLOCK(sigmgr, sigblock2);
15         // code that may result in signal generation
16         fclose(fp);
17     END_SIGNAL_BLOCK(asigmgr, sigblock2);
18     if (sigmgr->sigHandled()) {
19         cout << "SIGUSR1 occurred." << endl;
20         fclose(fp);
21     }
22 }
```

Chapter 6

Input/Output

The *BiometricEvaluation::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.

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

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

Listing 6.1: Using a RecordStore

```
1  #include <iostream>
2  #include <be_io_dbrecstore.h>
3  int
4  main(int argc, char* argv[]) {
5
6      IO::DBRecordStore *rs;
7      try {
8          rs = new IO::DBRecordStore("myRecords", "My Record Store", "");
9      } catch (Error::Exception& e) {
10         cout << "Caught " << e.getInfo() << endl;
11         return (EXIT_FAILURE);
12     }
13     auto_ptr<IO::DBRecordStore> ars(rs);
14
15     try {
16         uint8_t *theData;
17
18         theData = getSomeData();
19         ars->insert("key1", theData);
20
21         theData = getSomeData();
22         ars->insert("key2", theData);
23
24     } catch (Error::Exception& e) {
25         cout << "Caught " << e.getInfo() << endl;
26         return (EXIT_FAILURE);
27     }
28
29     // Some more processing where new data for a key comes in ...
30     theData = getSomeData();
```

```
31      ars->replace("key1", theData);
32
33      // Obtain the data for all keys ...
34      string theKey;
35      while (true) {
36          uint64_t len = rs->sequence(theKey, theData);
37          cout << "Read data for key " << theKey << " of length " << len
                << endl;
38      }
39      // The data for the key is no longer needed ...
40      ars->remove("key1");
41  }
```

6.3 Logging

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

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

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

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

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

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

Listing 6.2: Using a LogSheet within a LogCabinet

```
1  #include <be_io_logcabinet.h>
2  using namespace BiometricEvaluation;
3  using namespace BiometricEvaluation::IO;
4
5  LogCabinet *lc;
```

```
6  try {
7      lc = new LogCabinet(lcname, "A Log Cabinet", "");
8  } catch (Error::ObjectExists &e) {
9      cout << "The Log Cabinet already exists." << endl;
10     return (-1);
11 } catch (Error::StrategyError& e) {
12     cout << "Caught " << e.getInfo() << endl;
13     return (-1);
14 }
15 auto_ptr<LogCabinet> alc(lc);
16 try {
17     ls = alc->newLogSheet(lsname, "Log Sheet in Cabinet");
18 } catch (Error::ObjectExists &e) {
19     cout << "The Log Sheet already exists." << endl;
20     return (-1);
21 } catch (Error::StrategyError& e) {
22     cout << "Caught " << e.getInfo() << endl;
23     return (-1);
24 }
25 ls->setAutoSync(true);  // Force write of every entry when finished
26 int i = ...
27 *ls << "Adding an integer value " << i << " to the log." << endl;
28 ls->newEntry();         // Forces the write of the current entry
29 .........
30 delete ls;
31 return;                 // The LogCabinet is destructed by the auto_ptr
```

6.4 Properties

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

Listing 6.3: Using a Properties Object

```
1  IO::Properties *props;
2  string fname = "test.prop";
3  try {
4      props = new IO::Properties(fname);
5  } catch (Error::StrategyError &e) {
6      cerr << "Caught " << e.getInfo()  << endl;
7      return;
8  } catch (Error::FileError& e) {
9      cerr << "A file error occurred: " << e.getInfo() << endl;
10     return;
11 }
12 props->setProperty("foo", "bar");
13 props->setProperty("theAnswer", "42");
14     :
15     :
16     :
17 try {
18     int64_t theAnswer = props->getProperty("theAnswer");
19     cout << "The answer is " << theAnswer << endl;
```

```
20  } catch (Error::ObjectDoesNotExist &e) {
21      cerr << "The answer is elusive." << endl;
22      return;
23  }
24  string fooProp = props->getProperty("foo");
25  cout << "Foo is set to " << fooProp << endl;
26      :
27      :
28      :
29  try {
30      props->removeProperty("foo");
31  } catch (Error::ObjectDoesNotExist &e) {
32      cerr << "Failed to remove property." << endl;
33  }
```

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.

Watch dog 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

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

Within the *Watchdog* header file, two macros are defined: `BEGIN_WATCHDOG_BLOCK()` and `END_WATCHDOG_BLOCK()`, each taking the *Watchdog* object and label as parameters. The label must be unique for each watch dog 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 on the facing page shows. In most cases, however, careful application design can remove the need for duplicate code. In the example, placing the Timer start()/stop() calls outside of the Watchdog block simplifies the coding, although the small amount of time for the Watchdog setup and tear down would be included in the time.

Chapter 8

Process Information

The Process package is a set of APIs used to gather information on a process, or to limit the capabilities of a process.

8.1 Process Statistics

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

Listing 8.1: Gathering Process Statistics

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

```
19          diff = systemend - systemstart;
20          cout << "System time elapsed is " << diff << endl;
21      } catch (Error::Exception) {
22          cout << "Caught " << e.getInfo() << endl;
23      }
24
25  }
```

In addition to using the Process API to gather statistics to be returned from the function call, the API provides a means to have a "standard" set of statistics logged either synchronously or asynchronously to a LogSheet (See Section 6.3 on page 19) contained within a LogCabinet. Applications can start and stop logging at will to this LogSheet. Post-postmortem analysis can then be done on the entries in the LogSheet. Listing 8.2 shows the use of logging.

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

```
Description: Statistics for test_be_process_statistics (PID 28370)
# Entry 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

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

```
25          delete logstats;
26          return (EXIT_FAILURE);
27      }
28
29      // Set up asynchronous logging, every second
30      try {
31          logstats->startAutoLogging(1);
32      } catch (Error::ObjectExists &e) {
33          cout << "Caught " << e.getInfo() << endl;
34          delete logstats;
35          return (EXIT_FAILURE);
36      }
37
38      // Do some other work
39
40      // Stop logging
41      logstats->stopAutoLogging();
42      delete logstats;
43  }
```

Chapter 9

System

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

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

Listing 9.1: Using the System CPU Count Information

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

```
23                 cout << "Running a single process only." << endl;
24         }
25
26         // processing done by parent ...
27 }
```

Chapter 10

Image

The Image package maintains the classes and other information related to images and image processing. Within the Biometric Evaluation Framework, many classes refer to images, such as when dealing with fingerprint data. Many biometric data standards supply the actual image in several forms, either compressed or not. 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 enumerations and types used to represent aspects of an image. Compression algorithm and resolution scale are examples of enumerations and represent some information about the image. Types include structures to contain a coordinate on the image, or the image size. The BECommon does not perform any translation (within the Image classes) of scale units, or sizing, however, as each set of attributes is copied directly from the image itself, or the biometric record that contains the image.

The data types containing size and other information are used in the Image classes, as well as other classes, such as finger views, because those classes manage information taken directly from the biometric record. In many cases, the source image attributes are part of the biometric record. Applications can compare those values against those directly from the Image object, as in most cases those values are taken directly from the underlying image data. See Chapter 14 on page 43 for more information on image-based biometric records.

The Image namespace contains all of the Image classes that are used to represent an image in raw or compressed form. These classes are described in the following sec-

tions.

10.2 The Image Class

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

- As-recorded format binary data
- Compression algorithm
- Decompressed ("raw") format binary data, grayscale/full color
- Depth
- Dimensions, width/height
- Resolution

A rudimentary implementation of generating a grayscale image is provided by the Image class. This implementation calculates the luminance value Y (of YCbCr) for each pixel of a color image. Image subclasses may implement their own grayscale conversion methods.

10.3 Raw Image

The Raw Image class represents a decompressed image. Raw Image has no special implementation or additional methods.

10.4 JPEG Image

The JPEG Image class represents an image encoded according to the JPEG image standard [7]. Decompression and grayscale conversion are accomplished via libjpeg [5].

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 Image includes a JPEG memory source manager implementation, but it is built only if a version of libjpeg older than 8.0 is detected.

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

10.5 JPEG2000 Image

The JPEG2000 Image class provides Image class functionality to JPEG 2000-encoded images [6]. 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) [9]. JPEG2000 Image also provides a static function to test whether or not an image appears to be JPEG 2000-encoded.

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

Errors within libopenjpeg will be caught and rethrown as Exceptions.

10.6 PNG Image

The PNG Image class represents an image encoded according to the PNG image standard [4]. Decompression is provided by libpng [12].

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

10.7 WSQ Image

Images encoded in the WSQ-image standard [14] are represented by the WSQ Image class. The WSQ decompressor found in NBIS [10] 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 NBIS libwsq will propagate through stderr and will not be rethrown as exceptions.

Chapter 11

Text

The Text package consists of utility functions for string and character-based information. Examples include removing whitespace and calculating a digest over a string. Listing 11.1 shows how to use the Text package's utility functions.

Listing 11.1: Using the Text Package

```
1  #include <iostream>
2  #include <vector>
3  #include <be_text.h>
4
5  using namespace BiometricEvaluation;
6  using namespace std;
7
8  int
9  main(int argc, char* argv[])
10 {
11     cout << "Text::removeLeadingTrailingWhitespace()" << endl;
12     string whitespace = "␣␣␣␣␣foo␣␣␣bar␣␣␣␣␣␣";
13     cout << "\"" << whitespace << "\"␣=␣\"";
14     Text::removeLeadingTrailingWhitespace(whitespace);
15     cout << "\"" << whitespace << "\"␣=␣\"" << endl;
16
17     string secret_str = "secret_file_name.wsq";
18     cout << "MD5␣(\"" << secret_str << "\")␣=␣" <<
           Text::digest(secret_str) << endl;
19 }
20
21     string split_str1 = "This␣is,␣a␣string,␣split␣on␣commas.";
22     string split_str2 = "Semicolons␣are␣bad␣form;␣avoid␣them.";
23     cout << "Split␣\"" << split_str1 << "\"␣on␣','" << endl;
24     vector<string> str1_components = Text::split(split_str1, ',');
25     for (int i = 0; i < str1_components.size(); i++)
26         cout << "\t*␣\"" << str1_components[i] << "\"" << endl;
27
28     cout << "Split␣\"" << split_str2 << "\"␣on␣';'" << endl;
29     vector<string> str2_components = Text::split(split_str2, ';');
30     for (int i = 0; i < str2_components.size(); i++)
```

```
31          cout << "\t*␣\"" << str2_components[i] << "\"" << endl;
32
33      cout << "Split␣\"" << split_str2 << "\"␣on␣'z'" << endl;
34      vector<string> failed_split = Text::split(split_str2, 'z');
35      for (int i = 0; i < failed_split.size(); i++)
36          cout << "\t*␣\"" << failed_split[i] << "\"" << endl;
37      cout << endl;
38
39      string path =
            "/this/portion/is/the/dirname/and_this_is_the_filename";
40      cout << "Path:␣" << path << endl;
41      cout << "Dirname:␣" << Text::dirname(path) << endl;
42      cout << "Filename:␣" << Text::filename(path) << endl;
```

The output from the program shown in Listing 11.1 on the previous page is shown below:

```
Text::removeLeadingTrailingWhitespace()
"     foo   bar      " = "foo   bar"

MD5 ("secret_file_name.wsq") = 169a337d3689cbcfe508778a89419fa6

Split "This is, a string, split on commas." on ','
* "This is"
* " a string"
* " split on commas."
Split "Semicolons are bad form; avoid them." on ';'
* "Semicolons are bad form"
* " avoid them."
Split "Semicolons are bad form; avoid them." on 'z'
* "Semicolons are bad form; avoid them."

Path: /this/portion/is/the/dirname/and_this_is_the_filename
Dirname: /this/portion/is/the/dirname
Filename: and_this_is_the_filename
```

Chapter 12

Feature

The Feature package contains those items that relate to the representation of biometric features, such as fingerprint minutiae, facial features (eyes, etc.), and related information. Objects of these class types are typically associated with View (Chapter 14 on page 43) or DataInterchange (Chapter 15 on page 47) objects. For example, a minutiae object is usually obtained from a finger view, which may have been obtained from a data interchange object representing an entire biometric record for an individual.

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

12.1 ANSI/NIST Features

The ANSI/NIST [3] standard defines several features represented as data elements within a record. Fingerprint and palm minutiae is contained within Type-9 record. The AN2K7Minutiae class, contained in the Feature package, represents a single Type-9 record. An object of this class can be constructed directly from a complete ANSI/NIST record. However, it is more common for an application to retrieve these objects from the AN2KView object defined in the Finger package (Chapter 13 on page 41).

Chapter 13

Finger

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

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

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

When representing the information about a finger (and fingerprint), the class in the Finger package implements the interface defined in the View package. A finger is a specific type of view in that it represents all the available information about the finger, including the source image, minutiae (often in several formats), as well as the capture data (date, location, etc.) Finger views are documented in Section 14.1 on page 43.

13.1 ANSI/NIST Minutiae Data Record

The *AN2KMinutiaeDataRecord* class represents all of the information taken from a ANSI/NIST Type-9 record. A Type-9 record may include minutiae data items in several

formats (standard and proprietary) and the impression type code.

Chapter 14

View

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

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

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

14.1 Finger Views

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

The BECommon provides several classes that are derived from a base View class, con-

tained within the Finger package. See Chapter 13 on page 41 for more information on the types associated with fingers and fingerprints. This section discusses finger views, the classes which are derived from the general View class. These subclasses represent specific biometric file types, such as ANSI/NIST or INCITS/M1. In the latter case, two files must be provided when constructing the object because INCITS finger image and finger minutiae records are defined in two separate standards.

14.1.1 ANSI/NIST Finger Views

Listing 14.1: Using an AN2K Finger View

```
1  #include <fstream>
2  #include <iostream>
3  #include <be_finger_an2kview_fixedres.h>
4  using namespace std;
5  using namespace BiometricEvaluation;
6
7  int
8  main(int argc, char* argv[]) {
9
10     Finger::AN2KViewFixedResolution *_an2kv
11     try {
12         _an2kv = new Finger::AN2KViewFixedResolution("type9-3.an2k",
13             TYPE_3_ID, 1);
14     } catch (Error::DataError &e) {
15         cerr << "Caught " << e.getInfo()  << endl;
16         return (EXIT_FAILURE);
17     } catch (Error::FileError& e) {
18         cerr << "A file error occurred: " << e.getInfo() << endl;
19         return (EXIT_FAILURE);
20     }
21     std::auto_ptr<Finger::AN2KView> an2kv(_an2kv);
22
23     cout << "Image resolution is " << an2kv->getImageResolution() <<
           endl;
24     cout << "Image size is " << an2kv->getImageSize() << endl;
25     cout << "Image depth is " << an2kv->getImageDepth() << endl;
26     cout << "Compression is " << an2kv->getCompressionAlgorithm() <<
           endl;
27     cout << "Scan resolution is " << an2kv->getScanResolution() <<
           endl;
28
29     // Save the finger image to a file.
30     tr1::shared_ptr<Image::Image> img = an2kv->getImage();
31     if (img.get() == NULL) {
32        cerr << "Image was not present." << endl;
33        return (EXIT_FAILURE);
34     }
35     string filename = "rawimg";
36     ofstream img_out(filename.c_str(), ofstream::binary);
37     img_out.write((char *)&(img->getRawData()[0]),
38         img->getRawData().size());
39     if (img_out.good())
40             cout << "\tFile: " << filename << endl;
41     else {
```

```
42          img_out.close();
43          cerr << "Error␣occurred␣when␣writing␣" << filename << endl;
44          return (EXIT_FAILURE);
45      }
46      img_out.close();
47
48      // Get the finger minutiae sets. AN2K records can have more than
            one
49      // set of minutiae for a finger.
50
51      vector<Finger::AN2KMinutiaeDataRecord> mindata =
            an2kv->getMinutiaeDataRecordSet();
52  }
```

14.1.2 ISO/INCITS Finger Views

Chapter 15

Data Interchange

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

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

15.1 ANSI/NIST Data Records

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

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

Listing 15.1 on the next page shows how an application can retrieve all finger captures (Type-4 records) from an ANSI/NIST record. Once the Views are retrieved, the

application obtains the set of minutiae records associated with that View.

Listing 15.1: Retrieving ANSI/NIST Records

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

int
main(int argc, char* argv[])
{
    /*
     * Call the constructor that will open an existing AN2K file and
     * retrieve the first finger capture (Type-14) record.
     */
    std::auto_ptr<Finger::AN2KViewCapture> an2kv;
    try {
        an2kv.reset(new Finger::AN2KViewCapture("type9-14.an2k", 1));
    } catch (Error::DataError &e) {
        cout << "Caught " << e.getInfo() << endl;
        return (EXIT_FAILURE);
    } catch (Error::FileError& e) {
        cout << "A file error occurred: " << e.getInfo() << endl;
        return (EXIT_FAILURE);
    }

    cout << "Get the set of minutiae data records: ";
    vector<Finger::AN2KMinutiaeDataRecord> records =
        an2kv->getMinutiaeDataRecordSet();
    cout << "There are " << records.size() << " minutiae records." <<
        endl;

    /*
     * Get the info from the first minutiae record in the View.
     */
    DataInterchange::AN2KMinutiaeDataRecord type9 = records[0];

    /*
     * Get the "standard" set of minutiae.
     */
    Feature::AN2K7Minutiae an2k7m = type9.getAN2K7Minutiae();

    /*
     * Obtain the minutiae points, ridge counts, cores, and deltas.
     */
    Feature::MinutiaPointSet mps;
    Feature::RidgeCountItemSet rcs;
    Feature::CorePointSet cps;
    Feature::DeltaPointSet dps;
    try {
        mps = an2k7m->getMinutiaPoints();
        rcs = an2k7m->getRidgeCountItems();
        cps = an2k7m->getCores();
        dps = an2k7m->getDeltas();

    } catch (Error::DataError &e) {
        cout << "Caught " << e.getInfo() << endl;
        return (EXIT_FAILURE);
```

```
54      }
55
56      cout << "There are " << mps.size() << " minutiae points:" << endl;
57      /*
58       * Print out the minutiae points.
59       */
60      for (int i = 0; i < mps.size(); i++) {
61          printf("(%u,%u,%u)\n", mps[i].coordinate.x,
                mps[i].coordinate.y,
62               mps[i].theta);
63      }
64      cout << "There are " << rcs.size() << " ridge counts:" << endl;
65      for (int i = 0; i < rcs.size(); i++) {
66          printf("(%u,%u,%u)\n", rcs[i].index_one, rcs[i].index_two,
67          rcs[i].count);
68      }
69      cout << "There are " << cps.size() << " cores." << endl;
70      cout << "There are " << dps.size() << " deltas." << endl;
71
72      cout << "Fingerprint Reader: " << endl;
73      try { cout << an2k7m->getOriginatingFingerprintReadingSystem() <<
            endl; }
74      catch (Error::ObjectDoesNotExist) { cout << "<Omitted>" << endl; }
75
76      cout << "Pattern (primary): " <<
77      Feature::AN2K7Minutiae::convertPatternClassification(
78      an2k7m->getPatternClassificationSet().at(0)) << endl;
79
80      return(EXIT_SUCCESS);
81  }
```

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

Listing 15.2: Retrieving ANSI/NIST Records

```
1   #include <be_io_recordstore.h>
2   #include <be_data_interchange_an2k.h>
3   using namespace BiometricEvaluation;
4
5   void
6   processImageData(uint8_t *buf, uint32_t size)
7   {
8       :
9       :
10      :
11      :
12  }
13
14  int
15  main(int argc, char* argv[]) {
16
17      std::tr1::shared_ptr<IO::RecordStore> rs;
18      try {
```

```
19          rs = IO::RecordStore::openRecordStore(rsname, datadir,
                IO::READONLY);
20      } catch (Error::Exception &e) {
21          cerr << "Could not open record store: " << e.getInfo() << endl;
22          return (EXIT_FAILURE);
23      }
24
25      /*
26       * Read some AN2K records and construct the View objects.
27       */
28      Utility::uint8Array data;
29      string key;
30      while (true) {          // Loop through all records in store
31          uint64_t rlen;
32          try {
33              rlen = rs->sequence(key, NULL);
34          } catch (Error::ObjectDoesNotExist &e) {
35              break;
36          } catch (Error::Exception &e) {
37              cout << "Failed sequence: " << e.getInfo() << endl;
38              return (EXIT_FAILURE);
39          }
40          data.resize(rlen);
41          try {
42              rs->read(key, data);
43              DataInterchange::AN2KRecord an2k(data);
44              std::vector<Finger::AN2KViewLatent> latents =
                    an2k.getFingerLatents();
45              for (int i = 0; i < latents.size(); i++) {
46                  tr1::shared_ptr<Image::Image> img =
                        latents[i].getImage();
47                  if (img != NULL) {
48                      cout << "\tCompression: " <<
                            img->getCompressionAlgorithm() << endl;
49                      cout << "\tDimensions: " << img->getDimensions()
                            << endl;
50                      cout << "\tResolution: " << img->getResolution()
                            << endl;
51                      cout << "\tDepth: " << img->getDepth() << endl;
52                      processImageData(img->getRawData(),
                            img->getRawData().size());
53                  }
54              }
55          } catch (Error::Exception &e) {
56              return (EXIT_FAILURE);
57          }
58      }
59      return(EXIT_SUCCESS);
60  }
```

15.2 INCITS Data Records

This INCITS class of data records covers all those record formats that are derived from the standards defined by the InterNational Committee for Information Technol-

ogy Standards [8]. These formats include the ANSI-2004 Finger Minutiae Record Format [1], the ISO equivalent [2], and other data formats, including finger images.

15.2.1 Finger Views

Within the BECommon, finger view objects (Section 14.1) can be created from a combination of finger minutiae and image records. However, it is not necessary to have both records in order to create the view because each record contains enough information to represent the finger (image size, for example). However, if a view is 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 15.3 shows an example of accessing the information in an ANSI-2004 Finger Minutiae Record by creating an ANSI2004View object from the record file.

Listing 15.3: INCITS Finger Views

```
1  #include <be_finger_ansi2004view.h>
2  using namespace BiometricEvaluation;
3
4  int
5  main(int argc, char* argv[])
6  {
7      /*
8       * Create a finger view from a file containing an ANSI-2004 finger
9       * minutiae record, and without the finger view record.
10      */
11
12     Finger::ANSI2004View fngv;
13     try {
14         fngv = Finger::ANSI2004View("fmr.ansi2004", "", 3);
15     } catch (Error::DataError &e) {
16         cout << "Caught " << e.getInfo() << endl;
17         exit (EXIT_FAILURE);
18     } catch (Error::FileError& e) {
19         cout << "A file error occurred: " << e.getInfo() << endl;
20         exit (EXIT_FAILURE);
21     }
22
23     /*
24      * The ANSI200xView implementation of the View::View interface.
25      */
26     cout << "Image resolution is " << fngv.getImageResolution() <<
           endl;
27     cout << "Image size is " << fngv.getImageSize() << endl;
28     cout << "Image depth is " << fngv.getImageDepth() << endl;
29     cout << "Compression is " << fngv.getCompressionAlgorithm() <<
           endl;
30     cout << "Scan resolution is " << fngv.getScanResolution() << endl;
31
32     /*
33      * Test the ANSI200xView implementation of the Finger::INCITSVIEW
34      * interface.
```

```
35          */
36          cout << "Finger position is " << fngv.getPosition() << endl;
37          cout << "Impression type is " << fngv.getImpressionType() << endl;
38          cout << "Quality is " << fngv.getQuality() << endl;
39          cout << "Eqpt ID is " << hex << showbase <<
                fngv.getCaptureEquipmentID() << endl;
40          cout << dec;
41
42          Feature::INCITSMinutiae fmd = fngv.getMinutiaeData();
43          cout << "Minutiae format is " << fmd.getFormat() << endl;
44          Feature::MinutiaPointSet mps = fmd.getMinutiaPoints();
45          cout << "There are " << mps.size() << " minutiae points:" << endl;
46          for (int i = 0; i < mps.size(); i++)
47              cout << mps[i];
48
49          Feature::RidgeCountItemSet rcs = fmd.getRidgeCountItems();
50          cout << "There are " << rcs.size() << " ridge count items:" <<
                endl;
51          for (int i = 0; i < rcs.size(); i++)
52              cout << "\t" << rcs[i];
53
54          Feature::CorePointSet cores = fmd.getCores();
55          cout << "There are " << cores.size() << " cores:" << endl;
56          for (int i = 0; i < cores.size(); i++)
57              cout << "\t" << cores[i];
58
59          Feature::DeltaPointSet deltas = fmd.getDeltas();
60          cout << "There are " << deltas.size() << " deltas:" << endl;
61          for (int i = 0; i < deltas.size(); i++)
62              cout << "\t" << deltas[i];
63
64          exit (EXIT_SUCCESS);
65      }
```

Bibliography

[1] *ANSI INCITS 378-2004: Finger Minutiae Format for Data Interchange*. ANSI/INCITS, 2004. 51

[2] *ISO/IEC 19794-2: Information technology - Biometric data interchange formats - Part 2: Finger minutiae data*. ISO/IEC, first edition, 2005. 51

[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. 39, 41, 47

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

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

[6] Joint Photographic Experts Group. JPEG2000 Image Standard, 1992. http://www.jpeg.org/jpeg2000/index.html. 35

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

[8] InterNational Committee for Information Technology Standards. http://www.incits.org. 51

[9] Communications and Remote Sensing Lab, Université catholique de Louvain. OpenJPEG Library, 2011. http://www.openjpeg.org/. 35

[10] NIST Biometric Image Software, 2011. http://www.nist.gov/itl/iad/ig/nbis.cfm. 9, 35

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

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

[13] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, special edition, 2000. 3, 11

[14] 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

Namespace Index

A.1 Namespace List

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

Appendix B

Class Index

B.1 Class Hierarchy

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

Appendix C

Class Index

C.1 Class List

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

Appendix D

Namespace Documentation

D.1 BiometricEvaluation::DataInterchange Namespace Reference

Data interchange formats, such as ANSI/NIST or INCITS records.

Classes

- class AN2KRecord

  A class to represent an entire ANSI/NIST record.

D.1.1 Detailed Description

Data interchange formats, such as ANSI/NIST or INCITS records. The DataInterchange package contains classes that represent entire biometric data records.

D.2 BiometricEvaluation::Error Namespace Reference

Exceptions, and other error handling.

Classes

- class Exception

  The parent class of all BiometricEvaluation exceptions.

- class FileError

  File error when opening, reading, writing, etc.

- class ParameterError

  An invalid parameter was passed to a constructor or method.

- class ConversionError

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

- class DataError

  Error when reading data from an external source.

- class MemoryError

  An error occurred when allocating an object.

- class ObjectExists

  The named object exists and will not be replaced.

- class ObjectDoesNotExist

  The named object does not exist.

- class ObjectIsOpen

  The object is already opened.

- class ObjectIsClosed

  The object is closed.

- class StrategyError

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

- class NotImplemented

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

- class SignalManager

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

Functions

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

D.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.

D.2.2 Function Documentation

D.2.2.1 string BiometricEvaluation::Error::errorStr ( )

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

Returns

The current error message specified by errno.

D.3 BiometricEvaluation::Finger Namespace Reference

Biometric information relating to finger images and derived information.

Classes

- class Position

  Finger position codes.

- class Impression

  Finger and palm impression types.

- class FingerImageCode
- class AN2KMinutiaeDataRecord

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

- class AN2KView

  A class to represent single finger view and derived information.

- class AN2KViewCapture

  Represents an ANSI/NIST variable-resolution finger image.

- class AN2KViewFixedResolution

  A class to represent single finger view and derived information.

- class AN2KViewLatent
- class AN2KViewVariableResolution

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

- class ANSI2004View

  A class to represent single finger view and derived information.

- class ANSI2007View

  A class to represent single finger view and derived information.

- class INCITSView

  A class to represent single finger view and derived information.

- class ISO2005View

  A class to represent single finger view and derived information.

Typedefs

- typedef std::vector< Position::Kind > **PositionSet**
- typedef std::map< Position::Kind, FingerImageCode::Kind > **PositionDescriptors**

Functions

- std::ostream & **operator**<< (std::ostream &, const Position::Kind &)
- std::ostream & **operator**<< (std::ostream &, const Impression::Kind &)
- std::ostream & **operator**<< (std::ostream &, const FingerImageCode::Kind &)

- std::ostream & operator<< (std::ostream &stream, const AN2KViewCapture::AmputatedBandaged::Kind &ab)

  Output stream overload for AmputatedBandaged::Kind.

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

  Output stream overload for FingerSegmentPosition.

- std::ostream & operator<< (std::ostream &stream, const AN2KViewVariableResolution::PrintPositionCoordinate &ppc)

  Output stream overload for PrintPositionCoordinate.

D.3.1 Detailed Description

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

D.3.2 Function Documentation

D.3.2.1 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.

D.4 BiometricEvaluation::Framework Namespace Reference

Information about the framework.

Functions

- unsigned int getMajorVersion ()

  Framework major version.

- unsigned int getMinorVersion ()

  Framework minor version.

- std::string getCompiler ()

  Compiler used to compile this framework.

- std::string getCompileDate ()

  Date when this framework was compiled.

- std::string getCompileTime ()

  Time when this framework was compiled.

- std::string getCompilerVersion ()

  Version string of compiler used to compile this framework.

D.4.1 Detailed Description

Information about the framework.

D.4.2 Function Documentation

D.4.2.1 unsigned int BiometricEvaluation::Framework::getMajorVersion ( )

Framework major version.

Returns

The major version number of the BiometricFramework

D.4.2.2 unsigned int BiometricEvaluation::Framework::getMinorVersion ( )

Framework minor version.

Returns

The minor version of the BiometricEvaluation framework.

D.4.2.3 std::string BiometricEvaluation::Framework::getCompiler ( )

Compiler used to compile this framework.

Returns

The name of the compiler used to compile this framework.

D.4.2.4 std::string BiometricEvaluation::Framework::getCompileDate ( )

Date when this framework was compiled.

Returns

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

D.4.2.5 std::string BiometricEvaluation::Framework::getCompileTime ( )

Time when this framework was compiled.

Returns

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

D.4.2.6 std::string BiometricEvaluation::Framework::getCompilerVersion ( )

Version string of compiler used to compile this framework.

Returns

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

D.5 BiometricEvaluation::Image Namespace Reference

Basic information relating to images.

Classes

- class CompressionAlgorithm

  Image compression algorithms.

- struct Coordinate

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

- struct Size

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

- struct Resolution

  A structure to represent the resolution of an image.

- class Image

  Represent attributes common to all images.

- class JPEG

  A JPEG-encoded image.

- class JPEG2000

  A JPEG-2000-encoded image.

- class JPEGL

  A Lossless JPEG-encoded image.

- class NetPBM

  A NetPBM-encoded image.

- class PNG

  A PNG-encoded image.

- class RawImage

  An image with no encoding or compression.

- class WSQ

  A WSQ-encoded image.

Typedefs

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

Functions

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

  Output stream overload for CoordinateSet.

- std::ostream & **operator**<< (std::ostream &, const Size &)
- std::ostream & **operator**<< (std::ostream &, const Resolution &)
- std::ostream & **operator**<< (std::ostream &stream, const Resolution::Kind &kind)
- float distance (const Coordinate &p1, const Coordinate &p2)

  Calculate the distance between two points.

D.5.1 Detailed Description

Basic information relating to images. Classes and methods for manipulating images.

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

D.5.2 Function Documentation

D.5.2.1 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.

D.5.2.2 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.

D.6 BiometricEvaluation::IO Namespace Reference

Input/Output functionality.

Namespaces

- namespace Utility

Classes

- struct ManifestEntry
- class ArchiveRecordStore

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

- class DBRecordStore

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

- class FileRecordStore
- class LogSheet

  A class to represent a single logging mechanism.

- class LogCabinet

- class Properties

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

- class RecordStore

  A class to represent a data storage mechanism.

Typedefs

- typedef map< string, ManifestEntry > ManifestMap
- typedef map< string, string > **PropertiesMap**

D.6.1 Detailed Description

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

D.6.2 Typedef Documentation

D.6.2.1 typedef map<string, ManifestEntry> BiometricEvaluation::IO::ManifestMap

Convenience typedef for storing the manifest

D.7 BiometricEvaluation::IO::Utility Namespace Reference

Functions

- void removeDirectory (const string &directory, const string &prefix) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t getFileSize (const string &pathname) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- bool fileExists (const string &pathname) throw (Error::StrategyError)
- bool **pathIsDirectory** (const string &pathname) throw (Error::StrategyError)
- bool validateRootName (const string &name)
- bool constructAndCheckPath (const string &name, const string &parentDir, string &fullPath)

- int makePath (const string &path, const mode_t mode)

  Create an entire directory tree.

- Memory::uint8Array readFile (const string &path, ios_base::openmode=ios_base::binary) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Read the contents of a file into a buffer.

- void writeFile (const uint8_t *data, const size_t size, const string &path, ios_base::openmode mode=ios_base::binary) throw (Error::ObjectExists, Error::StrategyError)

  Write the contents of a buffer to a file.

- void writeFile (const Memory::uint8Array data, const string &path, ios_base::openmode mode=ios_base::binary) throw (Error::ObjectExists, Error::StrategyError)

  Write the contents of a buffer to a file.

D.7.1 Detailed Description

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

D.7.2 Function Documentation

D.7.2.1 void BiometricEvaluation::IO::Utility::removeDirectory (const string & *directory*, const string & *prefix*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Remove a directory.

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.

D.7.2.2 uint64_t BiometricEvaluation::IO::Utility::getFileSize (const string & *pathname*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Get the size of a file.

Parameters

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

Returns

The file size.

Exceptions

Error::ObjectDoesNotExist The named directory does not exist.

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

D.7.2.3 bool BiometricEvaluation::IO::Utility::fileExists (const string & *pathname*) throw (Error::StrategyError)

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.

D.7.2.4 bool BiometricEvaluation::IO::Utility::validateRootName (const string & *name*)

Check whether or not a string is valid as a name for a rooted entity, such as a RecordStore or other type of container that is persistent within the file system. Notably, name cannot contain path name separators ('/' and '\') or begin with whitespace.

Parameters

`[in]` ***name*** The proposed name for the entity.

Returns

true if the name is acceptable, false otherwise.

D.7.2.5 bool BiometricEvaluation::IO::Utility::constructAndCheckPath (const string & *name,* const string & *parentDir,* string & *fullPath*)

Construct a full path for a rooted entity, and return true if that path exists; false otherwise.

Parameters

`[in]` ***name*** The proposed name for the entity; cannot be a pathname.

`[in]` ***parentDir*** The name of the directory to contain the entity.

`[out]` ***fullPath*** The complete path to the new entity, when when true is returned; ambiguous when false is returned.

Returns

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

D.7.2.6 int BiometricEvaluation::IO::Utility::makePath (const string & *path,* const mode_t *mode*)

Create an entire directory tree.

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

Parameters

`[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.

D.7.2.7 Memory::uint8Array BiometricEvaluation::IO::Utility::readFile (const string & *path,* ios_base::openmode `= ios_base::binary`) throw (Error::ObjectDoesNotExist, Error::StrategyError)

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.

D.7.2.8 void BiometricEvaluation::IO::Utility::writeFile (const uint8_t * *data*, const size_t *size*, const string & *path*, ios_base::openmode *mode* = `ios_base::binary`) throw (Error::ObjectExists, Error::StrategyError)

Write the contents of a buffer to a file.

Parameters

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 but truncate not set, or path exists and is a directory.

StrategyError An error occurred when using the underlying storage system.

D.7.2.9 void BiometricEvaluation::IO::Utility::writeFile (const Memory::uint8Array *data*, const string & *path*, ios_base::openmode *mode* = `ios_base::binary`) throw (Error::ObjectExists, Error::StrategyError)

Write the contents of a buffer to a file.

Parameters

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 but truncate not set, or path exists and is a directory.

StrategyError An error occurred when using the underlying storage system.

D.8 BiometricEvaluation::Memory Namespace Reference

Support for memory-related operations.

Classes

- class AutoArray

  A class to represent a C-style array with C++ features like iterators and benefits like knowledge of the size.

- class AutoBuffer
- class IndexedBuffer

  Manage a memory buffer with an index.

Typedefs

- typedef AutoArray< uint8_t > **uint8Array**
- typedef AutoArray< uint16_t > **uint16Array**
- typedef AutoArray< uint32_t > **uint32Array**

D.8.1 Detailed Description

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

D.9 BiometricEvaluation::Process Namespace Reference

Process information and controls.

Classes

- class Statistics

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

D.9.1 Detailed Description

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

D.10 BiometricEvaluation::System Namespace Reference

Operating system, hardware, etc.

Functions

- uint32_t getCPUCount () throw (Error::NotImplemented)

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

- uint64_t getRealMemorySize () throw (Error::NotImplemented)

  Obtain the amount of real memory in the system.

- double getLoadAverage () throw (Error::NotImplemented)

  Obtain the system load average for the last minute.

D.10.1 Detailed Description

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

D.10.2 Function Documentation

D.10.2.1 uint32_t BiometricEvaluation::System::getCPUCount ( ) throw (Error::NotImplemented)

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

Returns

The number of processing units.

Exceptions

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

D.10.2.2 uint64_t BiometricEvaluation::System::getRealMemorySize ( ) throw (Error::NotImplemented)

Obtain the amount of real memory in the system.

Returns

The real memory size, in kilobytes.

Exceptions

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

D.10.2.3 double BiometricEvaluation::System::getLoadAverage ( ) throw (Error::NotImplemented)

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.

D.11 BiometricEvaluation::Text Namespace Reference

Text processing for string objects.

Functions

- void removeLeadingTrailingWhitespace (string &s)

  Remove lead and trailing white space from a string object.

- string digest (const string &s, const string &digest="md5") throw (Error::StrategyError)

  Compute the digest of a string.

- string digest (const void *buffer, const size_t buffer_size, const string &digest="md5") throw (Error::StrategyError)

  Compute the digest of a memory buffer.

- vector< string > split (const string &str, const char delimiter)

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

- string filename (const string &path)

  Extract the filename portion of a pathname.

- string dirname (const string &path)

  Extract the directory part of a pathname.

D.11.1 Detailed Description

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

D.11.2 Function Documentation

D.11.2.1 string BiometricEvaluation::Text::digest (const string & *s*, const string & *digest* = `"md5"`) throw (Error::StrategyError)

Compute the digest of a string.

Parameters

`[in]` ***s*** The string of which a digest should be computed.

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

Returns

An ASCII representation of the hex digits composing the digest.

D.11.2.2 string BiometricEvaluation::Text::digest (const void * *buffer*, const size_t *buffer_size*, const string & *digest* = *"md5"*) throw (Error::StrategyError)

Compute the digest of a memory buffer.

Parameters

[in] ***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.

Returns

An ASCII representation of the hex digits composing the digest.

D.11.2.3 vector<string> BiometricEvaluation::Text::split (const string & *str*, const char *delimiter*)

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.

Returns

vector<string> Vector of tokens, in order of appearance

Note

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

D.11.2.4 string BiometricEvaluation::Text::filename (const string & *path*)

Extract the filename portion of a pathname.

Parameters

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

Returns

Filename portion of path.

D.11.2.5 string BiometricEvaluation::Text::dirname (const string & *path*)

Extract the directory part of a pathname.

Parameters

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

Returns

Directory portion of path.

D.12 BiometricEvaluation::Time Namespace Reference

Support for time and timers.

Classes

- class Timer

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

- class Watchdog

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

Functions

- string getCurrentTime ()

  Return the current time as a string.

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

Variables

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

D.12.1 Detailed Description

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

D.13 BiometricEvaluation::View Namespace Reference

View information.

Classes

- class AN2KView

  A class to represent single biometric view and derived information.

- class AN2KViewVariableResolution

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

- class View

  A class to represent single biometric element view.

Functions

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

  Output stream overload for DeviceMonitoringMode.

- std::ostream & operator<< (std::ostream &stream, const AN2KViewVariableResolution::AN2KQualityMetric &qm)

  Output stream overload for AN2KQualityMetric.

D.13.1 Detailed Description

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

D.13.2 Function Documentation

D.13.2.1 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 E

Class Documentation

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

Amputated or bandaged code.

`#include <be_finger_an2kview_capture.h>`

Public Types

- enum Kind { Amputated, Bandaged, NA }

E.1.1 Detailed Description

Amputated or bandaged code.

E.1.2 Member Enumeration Documentation

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

Enumerator:

Amputated Amputation

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

NA Optional field -- not specified

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

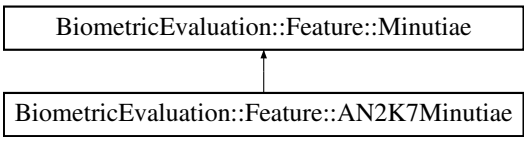
- be_finger_an2kview_capture.h

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

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

`#include <be_feature_an2k7minutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::AN2K7Minutiae:



Classes

- class EncodingMethod

  Methods for encoding minutiae data in an AN2K record.

- struct FingerprintReadingSystem

  Representation of information about a fingerprint reader system.

- class PatternClassification

  Pattern classification codes.

Public Types

- typedef std::vector< PatternClassification::Entry > **PatternClassificationSet**
- typedef struct FingerprintReadingSystem **FingerprintReadingSystem**

Public Member Functions

- AN2K7Minutiae (const std::string &filename, int recordNumber) throw (Error::DataError, Error::FileError)

  Construct an AN2K7 Minutiae object from file data.

- AN2K7Minutiae (const Memory::uint8Array &buf, int recordNumber) throw (Error::DataError)

  Construct an AN2K7 Minutiae object from data contained in a memory buffer.

- PatternClassificationSet getPatternClassificationSet () const

  Obtain the set fingerprint pattern classifications.

- FingerprintReadingSystem getOriginatingFingerprintReadingSystem () const throw (Error::ObjectDoesNotExist)
- MinutiaeFormat::Kind getFormat () const

  Obtain the minutiae format kind.

- MinutiaPointSet getMinutiaPoints () const

  Obtain the set of finger minutiae data points. The set may be empty.

- RidgeCountItemSet getRidgeCountItems () const

  Obtain the set of ridge count data items. The set may be empty.

- CorePointSet getCores () const

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

- DeltaPointSet getDeltas () const

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

Static Public Member Functions

- static PatternClassification::Kind convertPatternClassification (const char *fpc) throw (Error::DataError)

  Convert string read from AN2K record into a PatternClassification.

- static PatternClassification::Kind convertPatternClassification (const PatternClassification::Entry &entry) throw (Error::DataError)

  Convert a standard PatternClassification::Entry to a PatternClassification::Kind.

- static EncodingMethod::Kind convertEncodingMethod (const char *mem) throw (Error::DataError)

  Convert string read from AN2K record into a EncodingMethod.

- static Image::Coordinate convertCoordinate (const char *str, bool calculateDistance=true) throw (Error::DataError)

  Obtain a Coordinate given an AN2K entry.

E.2.1 Detailed Description

A class to represent a set of minutiae in an ANSI/NIST record. Each minutiae point, ridge count item, core, and delta is represented in the native ANSI/NIST format.

E.2.2 Constructor & Destructor Documentation

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

Construct an AN2K7 Minutiae object from file data.

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

Parameters

[in] ***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.

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

Construct an AN2K7 Minutiae object from data contained in a memory buffer.

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

Parameters

[in] ***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.

E.2.3 Member Function Documentation

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

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

Convert a standard PatternClassification::Entry to a PatternClassification::Kind.

Parameters

[in] ***entry*** A standard pattern classification entry

Exceptions

Error::DataError Non-standard pattern classification entry.

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

Convert string read from AN2K record into a EncodingMethod.

Parameters

[in] ***mem*** Value for minutiae encoding method read from AN2K record.

Exceptions

Error::DataError Invalid value for mem.

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

Obtain the set fingerprint pattern classifications.

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

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

Obtain the originating fingerprint reading system.

Exceptions

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

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

Obtain a Coordinate given an AN2K entry.

This AN2K entry is formatted as "XXXXYYYY".

Parameters

`[in]` ***str*** Coordinate string from an AN2K record.

`[in]` ***calculateDistance*** Whether or not to calculate the [xy]Distance portion of the Coordinate.

Returns

Image::Coordinate representation of str.

Exceptions

Error::DataError Invalid format of str.

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

- be_feature_an2k7minutiae.h

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

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

`#include <be_finger_an2kminutiae_data_record.h>`

Public Member Functions

- AN2KMinutiaeDataRecord (const string &filename, int recordNumber) throw (Error::DataError, Error::FileError)

  Construct an AN2KMinutiaeDataRecord object from data contained in a file on disk.

- AN2KMinutiaeDataRecord (const Memory::uint8Array &buf, int recordNumber) throw (Error::DataError)

  Construct an AN2KMinutiaeDataRecord object from data contained in a memory buffer.

- tr1::shared_ptr< Feature::AN2K7Minutiae > getAN2K7Minutiae () const

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

- Impression::Kind getImpressionType () const

  Return impression type field from Type-9 Record.

- map< uint16_t, Memory::uint8Array > getRegisteredVendorBlock (Feature::MinutiaeFormat::Kind vendor) const throw (Error::NotImplemented)

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

E.3.1 Detailed Description

Representation of a Type-9 Record from an AN2K file. Type-9 Records may contain only "standard" minutiae data (fields 9.005 - 9.012) or any combination of "standard" minutiae data and registered vendor minutiae data (several vendors from fields 9.013 - 9.175).

E.3.2 Constructor & Destructor Documentation

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

Construct an AN2KMinutiaeDataRecord object from data contained in a file on disk.

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

Parameters

[in] ***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.

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

Construct an AN2KMinutiaeDataRecord object from data contained in a memory buffer.

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

Parameters

[in] ***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.

E.3.3 Member Function Documentation

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

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

Returns

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

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

Return impression type field from Type-9 Record.

Returns

Impression type of the image from which minutiae points were generated.

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

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

Parameters

`[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.

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

- be_finger_an2kminutiae_data_record.h

E.4 BiometricEvaluation::View::AN2KViewVariableResolution::AN2KC Struct Reference

A structure to represent an AN2K quality metric.

`#include <be_view_an2kview_varres.h>`

Public Attributes

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

E.4.1 Detailed Description

A structure to represent an AN2K quality metric. The quality metric is an optional field in the Type-13 (Latent), Type-14 (Fingerprint and Segmentation) and Type-15 (Palmprint). The NIST Quality Metric is also returned via this structure.

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

- be_view_an2kview_varres.h

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

A class to represent an entire ANSI/NIST record.

`#include <be_data_interchange_an2k.h>`

Classes

- struct CharacterSet
- struct DomainName

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

Public Types

- typedef struct DomainName DomainName
- typedef struct CharacterSet CharacterSet

Public Member Functions

- AN2KRecord (const std::string filename) throw (Error::FileError, Error::DataError)

  Constructor taking an AN2K record from a file.

- AN2KRecord (const Memory::uint8Array &buf) throw (Error::DataError)

  Constructor taking an AN2K record from a buffer.

- string getVersionNumber () const

  Obtain the AN2K record version.

- string getDate () const

  Obtain the AN2K record date.

- string getDestinationAgency () const

  Obtain the destination agency ID.

- string getOriginatingAgency () const

  Obtain the originating agency ID.

- string getTransactionControlNumber () const

  Obtain the transcantion control number.

- string getNativeScanningResolution () const

  Obtain the native scanning resolution.

- string getNominalTransmittingResolution () const

  Obtain the nominal transmitting resolution.

- uint32_t getFingerLatentCount () const

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

- std::vector< Finger::AN2KViewLatent > getFingerLatents () const

  Obtain all latent (Type-13) finger views.

- uint32_t getFingerCaptureCount () const

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

- std::vector< Finger::AN2KViewCapture > getFingerCaptures () const

  Obtain all capture (Type-14) finger views.

- std::vector< Finger::AN2KMinutiaeDataRecord > getMinutiaeDataRecordSet () const

  Obtain all minutiae (Type-9) data.

- uint8_t getPriority () const

  Obtain the urgency with which a response is required.

- DomainName getDomainName () const

  Obtain the 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 set< int > recordLocations (const Memory::uint8Array &buf, int recordType) throw (Error::DataError)

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

- static set< int > recordLocations (const ANSI_NIST *an2k, int recordType)

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

E.5.1 Detailed Description

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

E.5.2 Member Typedef Documentation

E.5.2.1 typedef struct DomainName BiometricEvaluation::DataInterchange::AN2KRecord::DomainName

Convenience typedef for struct DomainName

E.5.2.2 typedef struct CharacterSet BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet

Convenience typedef for struct CharacterSet

E.5.3 Constructor & Destructor Documentation

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

Constructor taking an AN2K record from a file.

Parameters

[in] ***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.

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

Constructor taking an AN2K record from a buffer.

Parameters

[in] ***filename*** The memory buffer containing the complete ANSI/NIST record.

Exceptions

Error::DataError An error occurred when processing the AN2K record.

E.5.4 Member Function Documentation

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

E.5.4.2 static set<int> BiometricEvaluation::DataInterchange::AN2KRecord::recordLocations (const ANSI_NIST * *an2k*, int *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.

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

Obtain the AN2K record version.

Returns

The record version field in the Type-1 record.

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

Obtain the AN2K record date.

Returns

The date field in the Type-1 record.

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

Obtain the destination agency ID.

Returns

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

Obtain the originating agency ID.

Returns

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

Obtain the transcantion control number.

Returns

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

Obtain the native scanning resolution.

Returns

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

Obtain the nominal transmitting resolution.

Returns

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

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

Returns

The number of latent finger views in the AN2K record.

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

Obtain all latent (Type-13) finger views.

The returned vector will be empty when no latent views are present in the AN2KRecord.

Returns

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

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

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

Returns

The number of latent finger views in the AN2K record.

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

Obtain all capture (Type-14) finger views.

The returned vector will be empty when no capture views are present in the AN2KRecord.

Returns

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

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

Obtain all minutiae (Type-9) data.

Returns

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

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

Obtain the urgency with which a response is required.

Returns

Priority (1:High - 9:Low)

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

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

Returns

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

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

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

Returns

struct tm encoding of the GMT field.

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

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

Returns

Vector of CharacterSet structs representing other character sets that may appear in the transaction.

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

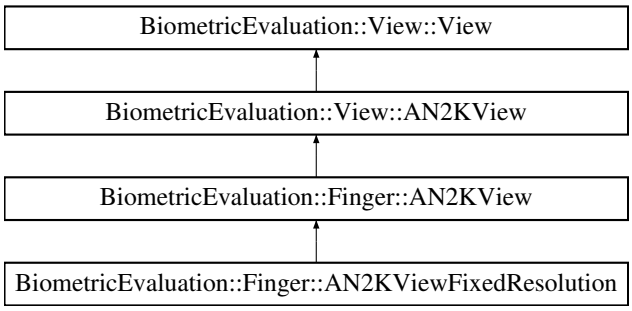
- be_data_interchange_an2k.h

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

A class to represent single finger view and derived information.

`#include <be_finger_an2kview.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KView:



Public Member Functions

- vector< AN2KMinutiaeDataRecord > getMinutiaeDataRecordSet () const throw (Error::DataError)

  Obtain the set of minutiae records.

- Finger::PositionSet getPositions () const

  Obtain the set of finger positions.

- Finger::Impression::Kind getImpressionType () const

  Obtain the finger impression code.

Static Public Member Functions

- static Finger::Position::Kind convertPosition (int an2kFGP) throw (Error::DataError)

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

- static Finger::PositionSet populateFGP (FIELD *field) throw (Error::DataError)

  Read the finger positions from an AN2K record.

- static Finger::Impression::Kind convertImpression (const unsigned char *str) throw (Error::DataError)

  Convert an impression code from a string.

- static Finger::FingerImageCode::Kind convertFingerImageCode (const char *str) throw (Error::DataError)

  Convert an finger image code from a string.

Protected Member Functions

- AN2KView (const std::string filename, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

  Construct an AN2K finger view from a file.

- **AN2KView** (const Memory::uint8Array &buf, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)
- void addMinutiaeDataRecord (Finger::AN2KMinutiaeDataRecord &mdr)

  Mutator for the AN2KMinutiaeDataRecord set.

- void setPositions (Finger::PositionSet &ps)

  Mutator for the position set.

- void setImpressionType (Finger::Impression::Kind &imp)

  Mutator for the impression type.

E.6.1 Detailed Description

A class to represent single finger view and derived information. A base Finger::AN2KView object represents an ANSI/NIST Type-3/4/5/6 record, and can return the image as well as the other information associated with that image, such as the minutiae from the corresponding Type-9 record.

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

E.6.2 Constructor & Destructor Documentation

E.6.2.1 BiometricEvaluation::Finger::AN2KView::AN2KView (const std::string *filename*, const uint8_t *typeID*, const uint32_t *recordNumber*) throw (Error::ParameterError, Error::DataError, Error::FileError) [protected]

Construct an AN2K finger view from a file.

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

Reimplemented from BiometricEvaluation::View::AN2KView.

E.6.3 Member Function Documentation

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

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

Parameters

[in] ***an2kFGP*** A finger position code as defined by the AN2K standard.

Exceptions

Error::DataError The position code is invalid.

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

Read the finger positions from an AN2K record.

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

Exceptions

Error::DataError The data contains an invalid value.

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

Convert an finger image code from a string.

Parameters

[in] ***str*** The character string containing the image code.

Returns

A FingerImageCode value.

Exceptions

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

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

Obtain the set of minutiae records.

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

Reimplemented from BiometricEvaluation::View::AN2KView.

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

Obtain the set of finger positions.

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

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

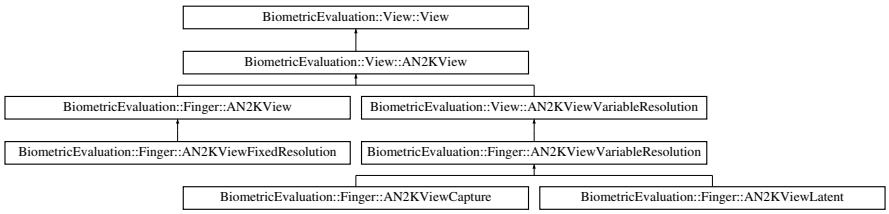
- be_finger_an2kview.h

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

A class to represent single biometric view and derived information.

`#include <be_view_an2kview.h>`

Inheritance diagram for BiometricEvaluation::View::AN2KView:



Classes

- class DeviceMonitoringMode

  The level of human monitoring for the image capture device.

Public Member Functions

- AN2KView (const std::string filename, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

  Construct an AN2K view from a file.

- **AN2KView** (const Memory::uint8Array &buf, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)
- tr1::shared_ptr< Image::Image > getImage () const

  Obtain the image used for the finger view.

- Image::Size getImageSize () const

  Obtain the image size.

- Image::Resolution getImageResolution () const

  Obtain the image resolution.

- uint32_t getImageDepth () const

  Obtain the image depth.

- Image::CompressionAlgorithm::Kind getCompressionAlgorithm () const

  Obtain the compression algorithm used on the image.

- Image::Resolution getScanResolution () const

  Obtain the image scan resolution.

- vector< Finger::AN2KMinutiaeDataRecord > getMinutiaeDataRecordSet () const throw (Error::DataError)

  Obtain the set of minutiae records.

Static Public Member Functions

- static DeviceMonitoringMode::Kind convertDeviceMonitoringMode (const char *dmm) throw (Error::DataError)

  Convert a device monitoring mode indicator from an AN2K record.

- static Image::CompressionAlgorithm::Kind convertCompressionAlgorithm (int recordType, const unsigned char *an2kValue) throw (Error::ParameterError, Error::DataError)

Static Public Attributes

- static const double MinimumScanResolutionPPMM

  Constants to define the minimum resolution used for fingerprint images in an AN2k record.

- static const double **HalfMinimumScanResolutionPPMM**
- static const int FixedResolutionBitDepth = 8

  The defined bit-depth for fixed-resolution images.

Protected Member Functions

- Memory::AutoBuffer< ANSI_NIST > getAN2K () const

  Obtain the complete ANSI/NIST record set.

- Memory::AutoArray< RECORD > getAN2KRecord () const

  Obtain the single ANSI/NIST record.

- void setImageData (const Memory::AutoArray< uint8_t > &imageData)

  Mutator for the image data.

- void setImageResolution (const Image::Resolution &ir)

  Mutator for the image resolution.

- void setImageDepth (const uint32_t depth)

  Mutator for the image depth.

- void setScanResolution (const Image::Resolution &ir)

  Mutator for the scan resolution.

- void setCompressionAlgorithm (const Image::CompressionAlgorithm::Kind &ca)

  Mutator for the compression algorithm.

E.7.1 Detailed Description

A class to represent single biometric view and derived information. This abstraction represents the image and derived information taken from an ANSI/NIST record.

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

E.7.2 Constructor & Destructor Documentation

E.7.2.1 BiometricEvaluation::View::AN2KView::AN2KView (const std::string *filename*, const uint8_t *typeID*, const uint32_t *recordNumber*) throw (Error::ParameterError, Error::DataError, Error::FileError)

Construct an AN2K view from a file.

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

Reimplemented in BiometricEvaluation::Finger::AN2KView.

E.7.3 Member Function Documentation

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

Convert a device monitoring mode indicator from an AN2K record.

Parameters

dmm Item value for device monitoring mode from an AN2K record.

Returns

DeviceMonitoringMode representation of dmm.

Exceptions

Error::DataError Invalid format of dmm.

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

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

Exceptions

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

Obtain the image used for the finger view.

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

Implements BiometricEvaluation::View::View.

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

Obtain the image size.

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

Implements BiometricEvaluation::View::View.

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

Obtain the image resolution.

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

Implements BiometricEvaluation::View::View.

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

Obtain the image depth.

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

Implements BiometricEvaluation::View::View.

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

Obtain the compression algorithm used on the image.

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

Implements BiometricEvaluation::View::View.

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

Obtain the image scan resolution.

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

Implements BiometricEvaluation::View::View.

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

Obtain the set of minutiae records.

Each AN2KViewVariableResolution may have more than one associated Type-9 record and each Type-9 record may have more than one minutiae format.

Reimplemented in BiometricEvaluation::Finger::AN2KView.

E.7.3.10 Memory::AutoArray<RECORD> BiometricEvaluation::View::AN2KView::getAN2KRecord ( ) const [protected]

Obtain the single ANSI/NIST record.

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

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

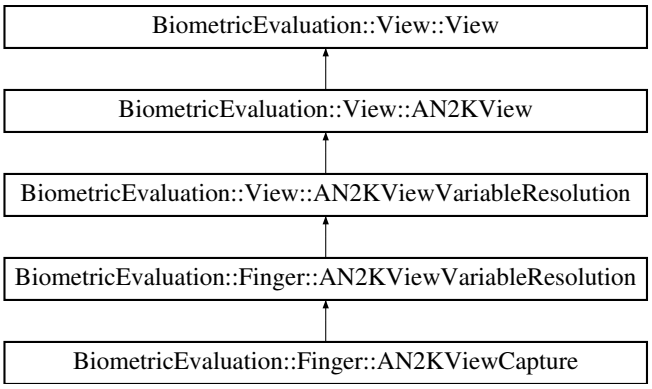
- be_view_an2kview.h

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

Represents an ANSI/NIST variable-resolution finger image.

`#include <be_finger_an2kview_capture.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewCapture:



Classes

- class AmputatedBandaged

  Amputated or bandaged code.

- struct FingerSegmentPosition

  Locations of an individual finger segment in a slap.

Public Types

- typedef struct FingerSegmentPosition **FingerSegmentPosition**
- typedef std::vector< FingerSegmentPosition > **FingerSegmentPositionSet**

Public Member Functions

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

  Construct an AN2K finger view from a file.

- AN2KViewCapture (const Memory::uint8Array &buf, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

  Construct an AN2K finger view using from a memory buffer.

- QualityMetricSet extractNISTQuality (const FIELD *field) throw (Error::DataError)

  Extract the NQM information from an AN2K FIELD.

- QualityMetricSet getNISTQualityMetric () const

  Obtain the NIST quality metric for all segmented finger images.

- QualityMetricSet getSegmentationQualityMetric () const
- AmputatedBandaged::Kind getAmputatedBandaged () const
- FingerSegmentPositionSet getFingerSegmentPositionSet () const
- FingerSegmentPositionSet getAlternateFingerSegmentPositionSet () const
- QualityMetricSet getFingerprintQualityMetric () const

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

Static Public Member Functions

- static AmputatedBandaged::Kind convertAmputatedBandaged (const char *ampcd) throw (Error::DataError)

  Convert string read from AN2K record into a AmputatedBandaged code.

- static FingerSegmentPosition convertFingerSegmentPosition (const SUBFIELD *sf) throw (Error::DataError)

  Convert SUBFIELD read from AN2K record into a FingerSegmentPosition struct.

- static FingerSegmentPosition convertAlternateFingerSegmentPosition (const SUBFIELD *sf) throw (Error::DataError)

  Convert SUBFIELD read from AN2K record into an AlternateFingerSegmentPosition struct.

E.8.1 Detailed Description

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

E.8.2 Constructor & Destructor Documentation

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

Construct an AN2K finger view from a file.

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

Parameters

`[in]` ***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.

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

Construct an AN2K finger view using from a memory buffer.

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

E.8.3 Member Function Documentation

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

Convert string read from AN2K record into a AmputatedBandaged code.

Parameters

[in] ***ampcd*** Value for amputated bandaged code read from an AN2K record.

Exceptions

Error::DataError Invalid value for ampcd.

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

Convert SUBFIELD read from AN2K record into a FingerSegmentPosition struct.

Parameters

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

Exceptions

Error::DataError Invalid value within sf.

E.8.3.3 static FingerSegmentPosition BiometricEvaluation::Finger::AN2KViewCapture::convertAlternateFingerSegmentPosition (const SUBFIELD * *sf*) throw (Error::DataError) [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.

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

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.

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

Obtain the NIST quality metric for all segmented finger images.

Returns

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

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

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

Obtain the segmentation quality metric for all segmented finger images.

Returns

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

E.8.3.7 AmputatedBandaged::Kind BiometricEvaluation::Finger::AN2KViewCapture::getAmputatedBandaged ( ) const

Returns

Optional amputated or bandaged code.

E.8.3.8 FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getFingerSegmentPositionSet ( ) const

Returns

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

E.8.3.9 FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getAlternateFingerSegmentPositionSet ( ) const

Returns

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

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

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

Returns

Fingerprint quality metrics

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

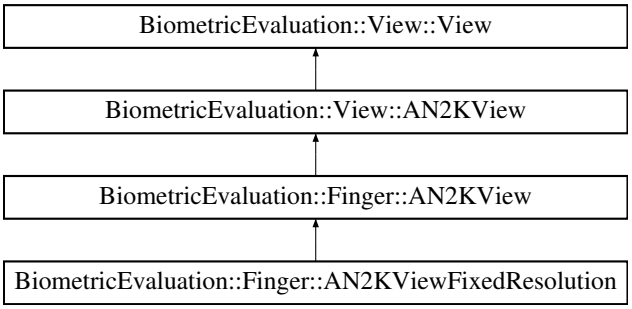
- be_finger_an2kview_capture.h

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

A class to represent single finger view and derived information.

`#include <be_finger_an2kview_fixedres.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewFixedResolution:



Public Member Functions

- AN2KViewFixedResolution (const std::string filename, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

  Construct an AN2K finger view from a file.

- **AN2KViewFixedResolution** (const Memory::uint8Array &buf, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

E.9.1 Detailed Description

A class to represent single finger view and derived information. A base Finger::AN2KView object represents an ANSI/NIST Type-3/4/5/6 record, and can return the image as well as the other information associated with that image, such as the minutiae from the corresponding Type-9 record.

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

E.9.2 Constructor & Destructor Documentation

E.9.2.1 BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution (const std::string *filename,* const uint8_t *typeID,* const uint32_t *recordNumber*) throw (Error::ParameterError, Error::DataError, Error::FileError)

Construct an AN2K finger view from a file.

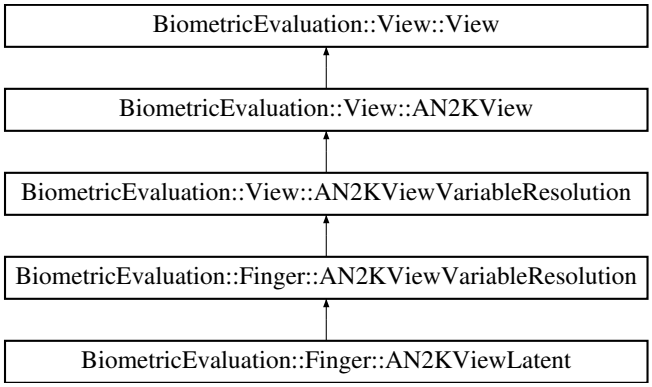
The file must contain the entire AN2K record, not just the finger image and/or minutiae records.

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

- be_finger_an2kview_fixedres.h

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

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewLatent:



Public Member Functions

- AN2KViewLatent (const std::string &filename, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

  Construct an AN2K finger view from a file.

- AN2KViewLatent (const Memory::uint8Array &buf, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

Construct an AN2K finger view using from a memory buffer.

- QualityMetricSet getLatentQualityMetric () const

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

- PositionDescriptors getSearchPositionDescriptors () const

  Return search position descriptors.

E.10.1 Constructor & Destructor Documentation

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

Construct an AN2K finger view from a file.

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

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

Construct an AN2K finger view using from a memory buffer.

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

E.10.2 Member Function Documentation

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

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

Returns

Latent quality metrics

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

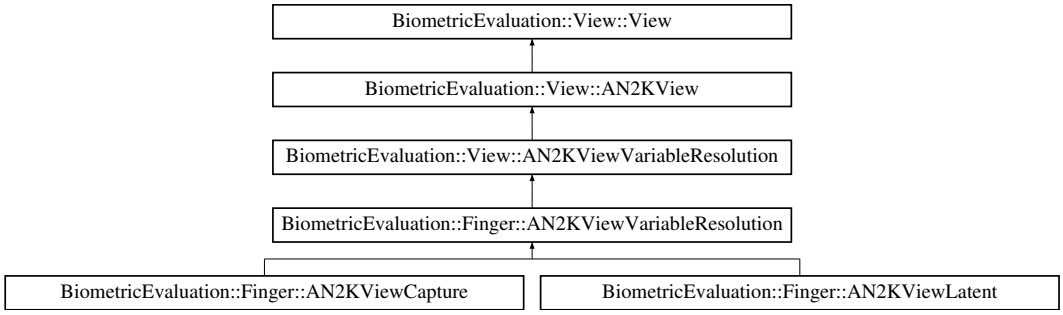
- be_finger_an2kview_latent.h

E.11 BiometricEvaluation::View::AN2KViewVariableResolution Class Reference

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

`#include <be_view_an2kview_varres.h>`

Inheritance diagram for BiometricEvaluation::View::AN2KViewVariableResolution:



Classes

- struct AN2KQualityMetric

  A structure to represent an AN2K quality metric.

Public Types

- typedef struct AN2KQualityMetric **AN2KQualityMetric**
- typedef std::vector< AN2KQualityMetric > **QualityMetricSet**

Public Member Functions

- string getSourceAgency () const

  Obtain the source agency.

- string getCaptureDate () const

  Obtain the capture date.

- string getComment () const

  Obtain the comment field.

- Memory::uint8Array getUserDefinedField (const uint16_t field) const throw (Error::ParameterError)

  Obtain a user-defined field.

Static Public Member Functions

- static QualityMetricSet extractQuality (FIELD *field) throw (Error::DataError)

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

- static Memory::uint8Array parseUserDefinedField (const Memory::AutoArray< RECORD > &record, int fieldID) throw (Error::ParameterError)

  Read raw bytes from a user-defined AN2K field.

Protected Member Functions

- AN2KViewVariableResolution (const std::string &filename, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

  Construct an AN2K finger view from a file.

- AN2KViewVariableResolution (const Memory::uint8Array &buf, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

  Construct an AN2K finger view using from a memory buffer.

- QualityMetricSet getQualityMetric () const

  Obtain quality metrics for associated image record.

E.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.

E.11.2 Constructor & Destructor Documentation

E.11.2.1 BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution (const std::string & *filename,* const uint8_t *typeID,* const uint32_t *recordNumber*) throw (Error::ParameterError, Error::DataError, Error::FileError) [protected]

Construct an AN2K finger view from a file.

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

Reimplemented in BiometricEvaluation::Finger::AN2KViewVariableResolution.

E.11.2.2 BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution (const Memory::uint8Array & *buf,* const uint8_t *typeID,* const uint32_t *recordNumber*) throw (Error::ParameterError, Error::DataError) [protected]

Construct an AN2K finger view using from a memory buffer.

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

Reimplemented in BiometricEvaluation::Finger::AN2KViewVariableResolution.

E.11.3 Member Function Documentation

E.11.3.1 static QualityMetricSet BiometricEvaluation::View::AN2KViewVariableResolution::extractQuality (FIELD * *field*) throw (Error::DataError) [static]

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

Parameters

[in] ***field*** A pointer to the field within the AN2K record. Error::DataError The data contains an invalid value.

E.11.3.2 string BiometricEvaluation::View::AN2KViewVariableResolution::getComment ( ) const

Obtain the comment field.

The comment field is optional in an AN2K record.

Returns

The comment field, empty string if not present.

E.11.3.3 Memory::uint8Array BiometricEvaluation::View::AN2KViewVariableResolution::getUserDefinedField (const uint16_t *field*) const throw (Error::ParameterError)

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.

E.11.3.4 static Memory::uint8Array BiometricEvaluation::View::AN2KViewVariableResolution::parseUserDefinedField (const Memory::AutoArray< RECORD > & *record*, int *fieldID*) throw (Error::ParameterError) `[static]`

Read raw bytes from a user-defined AN2K field.

Parameters

`[in]` ***record*** Reference 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.

E.11.3.5 QualityMetricSet BiometricEvaluation::View::AN2KViewVariableResolution::getQualityMetric ( ) const [protected]

Obtain quality metrics for associated image record.

Returns

Quality metrics

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

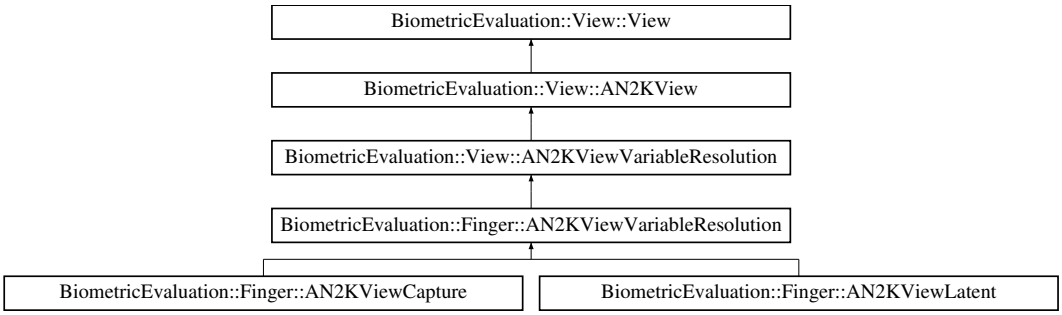
- be_view_an2kview_varres.h

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

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

`#include <be_finger_an2kview_varres.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewVariableResolution:



Classes

- struct PrintPositionCoordinate

  Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

Public Types

- typedef struct PrintPositionCoordinate **PrintPositionCoordinate**

- typedef std::vector< PrintPositionCoordinate > **PrintPositionCoordinateSet**

Public Member Functions

- Finger::PositionSet getPositions () const

  Obtain the set of finger positions.

- Finger::Impression::Kind getImpressionType () const

  Obtain the finger impression code.

- PositionDescriptors getPositionDescriptors () const

  Obtain the set of position descriptors.

- PrintPositionCoordinateSet getPrintPositionCoordinates () const

  Obtain print position coordinates.

Static Public Member Functions

- static PrintPositionCoordinate convertPrintPositionCoordinate (SUBFIELD *subfield) throw (Error::DataError)

  Convert a print position coordinate AN2K subfield to a PrintPositionCoordinate object.

- static PositionDescriptors parsePositionDescriptors (int typeID, const RECORD *record) throw (Error::DataError)

  Parse position descriptors from a record.

Protected Member Functions

- AN2KViewVariableResolution (const std::string &filename, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError, Error::FileError)

  Construct an AN2K finger view from a file.

- AN2KViewVariableResolution (const Memory::uint8Array &buf, const uint8_t typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

  Construct an AN2K finger view using from a memory buffer.

E.12.1 Detailed Description

A class to represent single finger view based on an ANSI/NIST record. The view represents a variable resolution (Type-13, 14) ANSI_NIST record.

E.12.2 Constructor & Destructor Documentation

E.12.2.1 BiometricEvaluation::Finger::AN2KViewVariableResolution::AN2KViewVariableResolution (const std::string & *filename,* const uint8_t *typeID,* const uint32_t *recordNumber*) throw (Error::ParameterError, Error::DataError, Error::FileError) [protected]

Construct an AN2K finger view from a file.

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

Reimplemented from BiometricEvaluation::View::AN2KViewVariableResolution.

E.12.2.2 BiometricEvaluation::Finger::AN2KViewVariableResolution::AN2KViewVariableResolution (const Memory::uint8Array & *buf,* const uint8_t *typeID,* const uint32_t *recordNumber*) throw (Error::ParameterError, Error::DataError) [protected]

Construct an AN2K finger view using from a memory buffer.

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

Reimplemented from BiometricEvaluation::View::AN2KViewVariableResolution.

E.12.3 Member Function Documentation

E.12.3.1 static PrintPositionCoordinate BiometricEvaluation::Finger::AN2KViewVariableResolution::convertPrintPositionCoordinate (SUBFIELD * *subfield*) throw (Error::DataError) [static]

Convert a print position coordinate AN2K subfield to a PrintPositionCoordinate object.

Parameters

[in] ***field*** A print position coordinate AN2K field

Returns

Object representation of field.

Exceptions

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

E.12.3.2 Finger::PositionSet BiometricEvaluation::Finger::AN2KViewVariableResolution::getPositions ( ) const

Obtain the set of finger positions.

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

E.12.3.3 static PositionDescriptors BiometricEvaluation::Finger::AN2KViewVariableResolution::parsePositionDescriptors (int *typeID*, const RECORD * *record*) throw (Error::DataError) [static]

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.

E.12.3.4 PrintPositionCoordinateSet BiometricEvaluation::Finger::AN2KViewVariableResolution::getPrintPositionCoordinates ( ) const

Obtain print position coordinates.

Returns

Set of all PrintPositionCoordinates

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

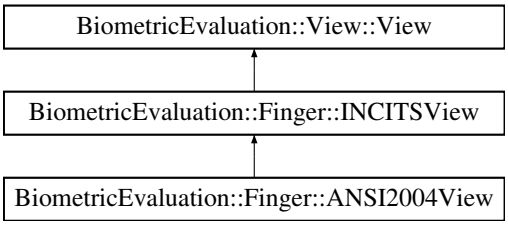
- be_finger_an2kview_varres.h

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

A class to represent single finger view and derived information.

`#include <be_finger_ansi2004view.h>`

Inheritance diagram for BiometricEvaluation::Finger::ANSI2004View:



Public Member Functions

- ANSI2004View ()

  Construct an empty ANSI finger view.

- ANSI2004View (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber) throw (Error::DataError, Error::FileError)

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

- ANSI2004View (const Memory::uint8Array &fmrBuffer, const Memory::uint8Array &firBuffer, const uint32_t viewNumber) throw (Error::DataError)

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

Static Public Attributes

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

- static const uint16_t **DELTA_X_COORD_MASK** = 0x3FFF
- static const uint16_t **DELTA_Y_COORD_MASK** = 0x3FFF

Protected Member Functions

- virtual void readCoreDeltaData (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas) throw (Error::DataError)

  Read the core points data.

E.13.1 Detailed Description

A class to represent single finger view and derived information. A Finger::ANSI2004View object represents a finger view from a INCITS/ANSI-2004 Finger Minutiae Record.

E.13.2 Constructor & Destructor Documentation

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

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

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

Parameters

[in] ***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.

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

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

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

Parameters

[in] ***fmrFilename*** The buffer containing the complete finger minutiae record.

[in] ***firFilename*** The buffer containing the complete finger image record.

[in] ***viewNumber*** The finger view number to use.

E.13.3 Member Function Documentation

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

Read the core points data.

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

Parameters

[in,out] ***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.

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

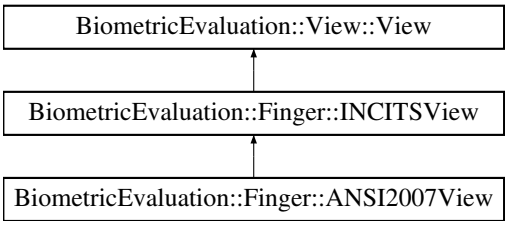
- be_finger_ansi2004view.h

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

A class to represent single finger view and derived information.

`#include <be_finger_ansi2007view.h>`

Inheritance diagram for BiometricEvaluation::Finger::ANSI2007View:



Public Member Functions

- ANSI2007View (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber) throw (Error::DataError, Error::FileError)

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

- ANSI2007View (const Memory::uint8Array &fmrBuffer, const Memory::uint8Array &firBuffer, const uint32_t viewNumber) throw (Error::DataError)

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

Static Public Attributes

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

Protected Member Functions

- void readFMRHeader (Memory::IndexedBuffer &buf, const uint32_t formatStandard) throw (Error::ParameterError, Error::DataError)

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

- void readFVMR (Memory::IndexedBuffer &buf) throw (Error::DataError)

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

- virtual void readCoreDeltaData (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas) throw (Error::DataError)

  Read the core points data.

E.14.1 Detailed Description

A class to represent single finger view and derived information. A Finger::ANSI2007View object represents a finger view from a INCITS/ANSI-2007 Finger Minutiae Record.

E.14.2 Constructor & Destructor Documentation

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

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

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

Parameters

[in] ***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.

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

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

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

Parameters

[in] ***fmrFilename*** The buffer containing the complete finger minutiae record.

[in] ***firFilename*** The buffer containing the complete finger image record.

[in] ***viewNumber*** The finger view number to use.

E.14.3 Member Function Documentation

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

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

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

Parameters

[in] ***buf*** The indexed buffer containing the record data. 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 FMR_ANSI2004_STANDARD or FMR_ISO2005_STANDARD.

Exceptions

ParameterError The specVersion parameter is incorrect.

DataError The INCITS record has invalid or missing data.

Reimplemented from BiometricEvaluation::Finger::INCITSView.

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

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

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

Parameters

[in,out] ***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.

Reimplemented from BiometricEvaluation::Finger::INCITSView.

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

Read the core points data.

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

Parameters

[in,out] ***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.

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

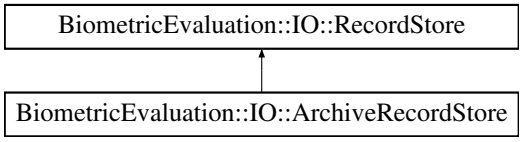
- be_finger_ansi2007view.h

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

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

`#include <be_io_archiverecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ArchiveRecordStore:



Public Member Functions

- ArchiveRecordStore (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- ArchiveRecordStore (const string &name, const string &parentDir, uint8_t mode=IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- ~ArchiveRecordStore ()
- uint64_t getSpaceUsed () throw (Error::StrategyError)

  Obtain real storage utilization.

- void sync () throw (Error::StrategyError)
- void insert (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
- void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t read (const string &key, void *const data) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void replace (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t length (const string &key) throw (Error::ObjectDoesNotExist)
- void flush (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t sequence (string &key, void *const data, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- bool needsVacuum ()
- string getArchiveName () const
- string getManifestName () const

Static Public Member Functions

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

E.15.1 Detailed Description

This class implements the IO::RecordStore interface by storing data items in single file, with an associated manifest file. Archives consist of binary records written back to back of each other. To pull information out of an archive, a manifest file is written in the same directory as the archive file.

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

key offset size

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

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

E.15.2 Constructor & Destructor Documentation

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

Create a new ArchiveRecordStore, read/write mode.

Parameters

`[in]` ***name*** The name of the store.

`[in]` ***description*** The store's description.

`[in]` ***parentDir*** The directory where the store is to be created.

Exceptions

Error::ObjectExists The store already exists.

Error::StrategyError An error occurred when accessing the underlying file system.

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

Open an existing ArchiveRecordStore.

Parameters

[in] ***name*** The name of the store.

[in] ***parentDir*** The directory where the store is to be created.

[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.

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

Destructor.

E.15.3 Member Function Documentation

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

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.15.3.3 void BiometricEvaluation::IO::ArchiveRecordStore::insert (const string & *key*, const void *const *data*, const uint64_t *size*) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Insert a record into the store.

Parameters

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

[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.

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

Remove a record from the store.

Parameters

[in] ***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.

E.15.3.5 uint64_t BiometricEvaluation::IO::ArchiveRecordStore::read (const string & *key*, void *const *const *data*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

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

Parameters

[in] ***key*** The key of the record to be read.

[in] ***data*** Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.15.3.6 void BiometricEvaluation::IO::ArchiveRecordStore::replace (const string & *key*, const void *const *const *data*, const uint64_t *size*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Replace a complete record in a store.

Parameters

[in] ***key*** The key of the record to be replaced.

[in] ***data*** The data for the record.

[in] ***size*** The size of data.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

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

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

Commit the record's data to storage.

Parameters

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

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

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

Sequence through a RecordStore, returning the key/data pairs. Sequencing means to start at some point in the store and return the record, then repeatedly calling the sequencor to return the next record. The starting point is typically the the first record, and is

set to that when the RecordStore object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

`[out]` ***key*** The key of the currently sequenced record.

`[in]` ***data*** Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.

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

Returns

The length of the record currently in sequence.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.15.3.10 void BiometricEvaluation::IO::ArchiveRecordStore::setCursorAtKey (string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Set the sequence cursor to an arbitrary position within the RecordStore, starting at key. Key will be the first record returned from the next call to sequence().

Parameters

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

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.15.3.11 void BiometricEvaluation::IO::ArchiveRecordStore::changeName (const string & *name*) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Change the name of the RecordStore.

Parameters

[in] ***name*** The new name for the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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

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

Returns

true if vacuum() would be beneficial false otherwise

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

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

Parameters

[in] ***name*** The name of the existing RecordStore.

[in] ***parentDir*** Where, in the filesystem, the store is rooted.

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

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

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

Parameters

[in] ***name*** The name of the existing RecordStore.

[in] ***parentDir*** Where, in the file system, the store is rooted.

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.

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

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

Returns

Path to archive file.

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

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

Returns

Path to manifest file.

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

- be_io_archiverecstore.h

E.16 BiometricEvaluation::Memory::AutoArray< T > Class Template Reference

A class to represent a C-style array with C++ features like iterators and benefits like knowledge of the size.

`#include <be_memory_autoarray.h>`

Public Types

- typedef T value_type

  Convenience typedef for the templated type.

- typedef T * iterator

  Convenience typedef for a pointer to the templated type.

- typedef const T * const_iterator

  Convenience typedef for a pointer to a const templated type.

- typedef T & reference

  Convenience typedef for a reference to the templated type.

- typedef const T & const_reference

  Convenience typedef for a reference to a const templated type.

Public Member Functions

- operator T * ()

  Dereference operator overload.

- operator T * () const

  Const dereference operator overload.

- reference operator[ ] (ptrdiff_t i)

  Indexing operator overload.

- const_reference operator[ ] (ptrdiff_t i) const

  Const indexing operator overload.

- AutoArray & operator= (const AutoArray &other)

  Assignment operator overload performing a deep copy.

- const_reference at (size_t offset) throw (out_of_range)

  Offset into the AutoArray with bounds checking.

- const_reference at (size_t offset) const throw (out_of_range)

  Offset into the AutoArray with bounds checking.

- iterator begin ()

  Obtain an iterator to the beginning of the AutoArray.

- const_iterator begin () const

  Obtain an iterator to the beginning of the AutoArray.

- iterator end ()

  Obtain an iterator to the end of the AutoArray.

- const_iterator end () const

  Obtain an iterator to the end of the AutoArray.

- size_t size () const

  Obtain the number of elements allocated for this AutoArray.

- void resize (size_t new_size, bool free=false) throw (Error::StrategyError)

  Add/subtract the number of elements this AutoArray can hold.

- void copy (const T *buffer, size_t size)

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

- void copy (const T *buffer)

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

- AutoArray ()

  Construct an AutoArray.

- AutoArray (size_t size)

  Construct an AutoArray.

- AutoArray (const AutoArray &copy)

  Construct an AutoArray.

E.16.1 Detailed Description

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

A class to represent a C-style array with C++ features like iterators and benefits like knowledge of the size.

E.16.2 Constructor & Destructor Documentation

E.16.2.1 template<class T > BiometricEvaluation::Memory::AutoArray< T >::AutoArray ( )

Construct an AutoArray.

The AutoArray will be of size 0.

E.16.2.2 template<class T > BiometricEvaluation::Memory::AutoArray< T >::AutoArray (size_t *size*)

Construct an AutoArray.

Parameters

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

E.16.2.3 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.

E.16.3 Member Function Documentation

E.16.3.1 template<class T > BiometricEvaluation::Memory::AutoArray< T >::operator T * ( )

Dereference operator overload.

Resolves to a pointer to the beginning of the underlying array storage of the AutoArray.

E.16.3.2 template<class T > BiometricEvaluation::Memory::AutoArray< T >::operator T * ( ) const

Const dereference operator overload.

Resolves to a pointer to the beginning of the underlying array storage of the AutoArray.

E.16.3.3 template<class T > BiometricEvaluation::Memory::AutoArray< T >::reference BiometricEvaluation::Memory::AutoArray< T >::operator[ ] (ptrdiff_t *i*)

Indexing operator overload.

Parameters

`[in]` ***i*** Index

Returns

Reference to element at index i.

E.16.3.4 template<class T > BiometricEvaluation::Memory::AutoArray< T >::const_reference BiometricEvaluation::Memory::AutoArray< T >::operator[ ] (ptrdiff_t *i*) const

Const indexing operator overload.

Parameters

`[in]` ***i*** Index

Returns

Reference to const element at index i.

E.16.3.5 template<class T > BiometricEvaluation::Memory::AutoArray< T > & BiometricEvaluation::Memory::AutoArray< T >::operator= (const AutoArray< T > & *other*)

Assignment operator overload performing a deep copy.

Parameters

[in] ***other*** AutoArray to be copied

Returns

Reference to a new AutoArray object.

E.16.3.6 template<class T > BiometricEvaluation::Memory::AutoArray< T >::const_reference BiometricEvaluation::Memory::AutoArray< T >::at (size_t *offset*) throw (out_of_range)

Offset into the AutoArray with bounds checking.

Parameters

offset Index into the AutoArray.

Returns

Const reference to an element at offset.

Exceptions

out_of_range Offset offset is not valid given the size of this AutoArray.

E.16.3.7 template<class T > BiometricEvaluation::Memory::AutoArray< T >::const_reference BiometricEvaluation::Memory::AutoArray< T >::at (size_t *offset*) const throw (out_of_range)

Offset into the AutoArray with bounds checking.

Parameters

offset Index into the AutoArray.

Returns

Const reference to an element at offset.

Exceptions

out_of_range Offset offset is not valid given the size of this AutoArray.

E.16.3.8 template<class T > BiometricEvaluation::Memory::AutoArray< T >::iterator BiometricEvaluation::Memory::AutoArray< T >::begin ( )

Obtain an iterator to the beginning of the AutoArray.

Returns

Pointer to the first element of the AutoArray.

E.16.3.9 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

Pointer to the const first element of the AutoArray.

E.16.3.10 template<class T > BiometricEvaluation::Memory::AutoArray< T >::iterator BiometricEvaluation::Memory::AutoArray< T >::end ( )

Obtain an iterator to the end of the AutoArray.

Returns

Pointer to the const last element of the AutoArray.

E.16.3.11 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

Pointer to the const last element of the AutoArray.

E.16.3.12 template<class T > size_t BiometricEvaluation::Memory::AutoArray< T >::size ( ) const

Obtain the number of elements allocated for this AutoArray.

Returns

Number of allocated elements.

E.16.3.13 template<class T > void BiometricEvaluation::Memory::AutoArray< T >::resize (size_t *new_size*, bool *free* = `false`) throw (Error::StrategyError)

Add/subtract the number of elements this AutoArray can hold.

This method can grow or shrink the number of allocated elements.

Parameters

new_size The number of elements the AutoArray should have allocated.

free Whether or not excess memory should be freed, in the case that new_size is smaller than the current AutoArray size.

Exceptions

Error::StrategyError Problem allocating memory.

E.16.3.14 template<class T> void BiometricEvaluation::Memory::AutoArray< T >::copy (const T * *buffer*, size_t *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.

E.16.3.15 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.

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

- be_memory_autoarray.h

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

Public Types

- typedef T value_type

  Manage a memory buffer.

- typedef T & **reference**
- typedef const T & **const_reference**

Public Member Functions

- **operator T** * ()
- T * **operator->** ()
- AutoBuffer & **operator=** (const AutoBuffer &other)
- **AutoBuffer** (T *data)
- **AutoBuffer** (int(*ctor)(T **), void(*dtor)(T *), int(*copyCtor)(T **, T *)=NULL)
- **AutoBuffer** (const AutoBuffer &copy)

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

E.17.1 Member Typedef Documentation

E.17.1.1 template<class T> typedef T BiometricEvaluation::Memory::AutoBuffer< T >::value_type

Manage a memory buffer.

It's easier to think of AutoBuffer as a wrapper for a pointer rather than the object it truly is. Therefore, you can interact with the AutoBuffer object exactly how you would a traditional pointer, without worrying about memory management.

Say you wanted to use an ANSI_NIST* but didn't want to be responsible for allocating or freeing the memory. Create an AutoBuffer object like:

AutoBuffer<ANSI_NIST> obj = AutoBuffer(allocator_fn, deallocator_fn[, copy_-constructor]);

Notice the AutoBuffer is for ANSI_NIST and not ANSI_NIST*, since AutoBuffer will handle the pointer for you. You can pass the AutoBuffer<ANSI_NIST> object to any function that takes an ANSI_NIST*. For example, it's perfectly valid to pass our 'obj' object above to:

write_fmttext(FILE *, ANSI_NIST *)

If you want to access a member from 'obj', you can use the dereference operator just like you would on a regular ANSI_NIST*:

int size = obj->num_bytes;

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

- be_memory_autobuffer.h

E.18 be_workorder Struct Reference

Public Attributes

- int **sockfd**
- void * **stateData**

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

- be_netsdk.h

E.19 BiometricEvaluation::DataInterchange::AN2KRecord::CharacterS Struct Reference

Public Member Functions

- CharacterSet (uint16_t identifier=0, string commonName="", string version="")

  Create a new CharacterSet struct.

Public Attributes

- uint16_t identifier
- string commonName
- string version

E.19.1 Constructor & Destructor Documentation

E.19.1.1 BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::CharacterSet (uint16_t *identifier* = 0, string *commonName* = "", 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.

E.19.2 Member Data Documentation

E.19.2.1 uint16_t BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::identifier

Identifier (000-999)

E.19.2.2 string BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::commonName

Common name of the character set

E.19.2.3 string BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::version

Optional version of the character set

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

- be_data_interchange_an2k.h

E.20 BiometricEvaluation::Image::CompressionAlgorithm Class Reference

Image compression algorithms.

`#include <be_image.h>`

Public Types

- enum **Kind** {
  None = 0, **Facsimile** = 1, **WSQ20** = 2, **JPEGB** = 3,
  JPEGL = 4, **JP2** = 5, **JP2L** = 6, **PNG** = 7,
  NetPBM = 8 }

E.20.1 Detailed Description

Image compression algorithms.

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

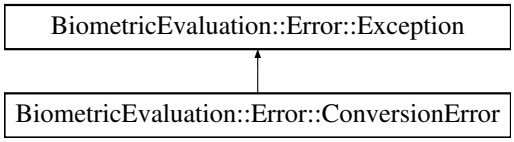
- be_image.h

E.21 BiometricEvaluation::Error::ConversionError Class Reference

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

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ConversionError:



Public Member Functions

- ConversionError ()
- ConversionError (string info)

E.21.1 Detailed Description

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

E.21.2 Constructor & Destructor Documentation

E.21.2.1 BiometricEvaluation::Error::ConversionError::ConversionError ( )

Construct a ConversionError object with the default information string.

Returns

The ConversionError object.

E.21.2.2 BiometricEvaluation::Error::ConversionError::ConversionError (string *info*)

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

Returns

The ConversionError object.

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

- be_error_exception.h

E.22 BiometricEvaluation::Image::Coordinate Struct Reference

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

`#include <be_image.h>`

Public Member Functions

- Coordinate (const uint32_t x=0, const uint32_t y=0, const float xDistance=0, const float yDistance=0)

  Create a Coordinate struct.

Public Attributes

- uint32_t x
- uint32_t y
- float xDistance
- float yDistance

E.22.1 Detailed Description

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

E.22.2 Constructor & Destructor Documentation

E.22.2.1 BiometricEvaluation::Image::Coordinate::Coordinate (const uint32_t *x = 0*, const uint32_t *y = 0*, const float *xDistance = 0*, const float *yDistance = 0*)

Create a Coordinate struct.

Parameters

`[in]` ***x*** X-coordinate

`[in]` ***y*** Y-coordinate

`[in]` ***xDistance*** X-coordinate distance from origin

`[in]` ***yDistance*** Y-coordinate distance from origin

E.22.3 Member Data Documentation

E.22.3.1 uint32_t BiometricEvaluation::Image::Coordinate::x

X-coordinate

E.22.3.2 uint32_t BiometricEvaluation::Image::Coordinate::y

Y-coordinate

E.22.3.3 float BiometricEvaluation::Image::Coordinate::xDistance

X-coordinate distance from origin

E.22.3.4 float BiometricEvaluation::Image::Coordinate::yDistance

Y-coordinate distance from origin

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

- be_image.h

E.23 BiometricEvaluation::Feature::CorePoint Struct Reference

Representation of the core.

`#include <be_feature_minutiae.h>`

Public Member Functions

- CorePoint (Image::Coordinate coordinate, bool has_angle=false, int angle=0)
  Create a CorePoint struct.

Public Attributes

- Image::Coordinate **coordinate**
- bool **has_angle**
- int **angle**

E.23.1 Detailed Description

Representation of the core. A core has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

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

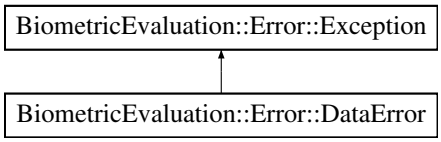
- be_feature_minutiae.h

E.24 BiometricEvaluation::Error::DataError Class Reference

Error when reading data from an external source.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::DataError:



Public Member Functions

- DataError ()
- DataError (string info)

E.24.1 Detailed Description

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

E.24.2 Constructor & Destructor Documentation

E.24.2.1 BiometricEvaluation::Error::DataError::DataError ( )

Construct a DataError object with the default information string.

Returns

The DataError object.

E.24.2.2 BiometricEvaluation::Error::DataError::DataError (string *info*)

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

Returns

The DataError object.

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

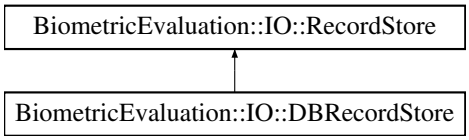
- be_error_exception.h

E.25 BiometricEvaluation::IO::DBRecordStore Class Reference

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

`#include <be_io_dbrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::DBRecordStore:



Public Member Functions

- DBRecordStore (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- DBRecordStore (const string &name, const string &parentDir, uint8_t mode=IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t getSpaceUsed () throw (Error::StrategyError)

  Obtain real storage utilization.

- void sync () throw (Error::StrategyError)
- void insert (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
- void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t read (const string &key, void *const data) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void replace (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t length (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void flush (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t sequence (string &key, void *const data, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)

E.25.1 Detailed Description

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

E.25.2 Constructor & Destructor Documentation

E.25.2.1 BiometricEvaluation::IO::DBRecordStore::DBRecordStore (const string & *name*, const string & *description*, const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new DBRecordStore, read/write mode.

Parameters

[in] ***name*** The name of the store.

[in] ***description*** The store's description.

[in] ***parentDir*** The directory where the store is to be created.

Exceptions

Error::ObjectExists The store already exists.

Error::StrategyError An error occurred when accessing the underlying file system.

E.25.2.2 BiometricEvaluation::IO::DBRecordStore::DBRecordStore (const string & *name*, const string & *parentDir*, uint8_t *mode* = *IO::READWRITE*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing DBRecordStore.

Parameters

[in] ***name*** The name of the store.

[in] ***parentDir*** The directory where the store is to be created.

[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.

E.25.3 Member Function Documentation

E.25.3.1 uint64_t BiometricEvaluation::IO::DBRecordStore::getSpaceUsed ( ) throw (Error::StrategyError) [virtual]

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.25.3.2 void BiometricEvaluation::IO::DBRecordStore::sync ( ) throw (Error::StrategyError) [virtual]

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.25.3.3 void BiometricEvaluation::IO::DBRecordStore::insert (const string & *key*, const void *const *data*, const uint64_t *size*) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Insert a record into the store.

Parameters

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

[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.

E.25.3.4 void BiometricEvaluation::IO::DBRecordStore::remove (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Remove a record from the store.

Parameters

[in] ***key*** The key of the record to be removed.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.25.3.5 uint64_t BiometricEvaluation::IO::DBRecordStore::read (const string & *key*, void *const *const *data*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

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

Parameters

[in] ***key*** The key of the record to be read.

[in] ***data*** Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.25.3.6 void BiometricEvaluation::IO::DBRecordStore::replace (const string & *key*, const void *const *const *data*, const uint64_t *size*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Replace a complete record in a store.

Parameters

[in] ***key*** The key of the record to be replaced.

[in] ***data*** The data for the record.

[in] ***size*** The size of data.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.25.3.7 uint64_t BiometricEvaluation::IO::DBRecordStore::length (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Return the length of a record.

Parameters

[in] ***key*** The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.25.3.8 void BiometricEvaluation::IO::DBRecordStore::flush (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Commit the record's data to storage.

Parameters

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

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.25.3.9 uint64_t BiometricEvaluation::IO::DBRecordStore::sequence (string & *key*, void *const *const *data*, int *cursor* = BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Sequence through a RecordStore, returning the key/data pairs. Sequencing means to start at some point in the store and return the record, then repeatedly calling the sequencor to return the next record. The starting point is typically the the first record, and is

set to that when the RecordStore object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

`[out]` ***key*** The key of the currently sequenced record.

`[in]` ***data*** Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.

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

Returns

The length of the record currently in sequence.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.25.3.10 void BiometricEvaluation::IO::DBRecordStore::setCursorAtKey (string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Set the sequence cursor to an arbitrary position within the RecordStore, starting at key. Key will be the first record returned from the next call to sequence().

Parameters

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

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.25.3.11 void BiometricEvaluation::IO::DBRecordStore::changeName (const string & *name*) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Change the name of the RecordStore.

Parameters

`[in]` ***name*** The new name for the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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

- be_io_dbrecstore.h

E.26 BiometricEvaluation::Feature::DeltaPoint Struct Reference

Representation of the delta.

`#include <be_feature_minutiae.h>`

Public Member Functions

- DeltaPoint (Image::Coordinate coordinate, bool has_angle=false, int angle1=0, int angle2=0, int angle3=0)

  Create a DeltaPoint struct.

Public Attributes

- Image::Coordinate **coordinate**
- bool **has_angle**
- int **angle1**
- int **angle2**
- int **angle3**

E.26.1 Detailed Description

Representation of the delta. A delta has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

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

- be_feature_minutiae.h

E.27 BiometricEvaluation::View::AN2KView::DeviceMonitoringMode Class Reference

The level of human monitoring for the image capture device.

`#include <be_view_an2kview.h>`

Public Types

- enum Kind {
  Controlled, Assisted, Observed, Unattended,
  Unknown, NA }

E.27.1 Detailed Description

The level of human monitoring for the image capture device.

E.27.2 Member Enumeration Documentation

E.27.2.1 enum BiometricEvaluation::View::AN2KView::DeviceMonitoringMode::Kind

Enumerator:

Controlled Operator physically controls the subject to acquire biometric sample.

Assisted Person available to provide assistance to the subject submitting the biometric.

Observed Person present to observe the operation of the device but provides no assistance.

Unattended No one present to observe or provide assistance.

Unknown No information is known.

NA Optional field -- not specified

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

- be_view_an2kview.h

E.28 BiometricEvaluation::DataInterchange::AN2KRecord::DomainName Struct Reference

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

`#include <be_data_interchange_an2k.h>`

Public Member Functions

- DomainName (string identifier="", string version="")

  Create a DomainName struct.

Public Attributes

- string identifier
- string version

E.28.1 Detailed Description

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

E.28.2 Constructor & Destructor Documentation

E.28.2.1 BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::DomainName (string *identifier* = "", 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.

E.28.3 Member Data Documentation

E.28.3.1 string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::identifier

Unique identifier for agency, entity, or implementation.

E.28.3.2 string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::version

Optional version of the implementation

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

- be_data_interchange_an2k.h

E.29 BiometricEvaluation::Feature::AN2K7Minutiae::EncodingMethod Class Reference

Methods for encoding minutiae data in an AN2K record.

`#include <be_feature_an2k7minutiae.h>`

Public Types

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

E.29.1 Detailed Description

Methods for encoding minutiae data in an AN2K record.

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

- be_feature_an2k7minutiae.h

E.30 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry Struct Reference

Public Member Functions

- Entry (bool standard, std::string code)

Public Attributes

- bool standard
- std::string code

E.30.1 Constructor & Destructor Documentation

E.30.1.1 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.

E.30.2 Member Data Documentation

E.30.2.1 bool BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::standard

Whether code is a standard AN2K pattern classification code.

E.30.2.2 std::string BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::code

AN2K or user-defined pattern classification code.

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

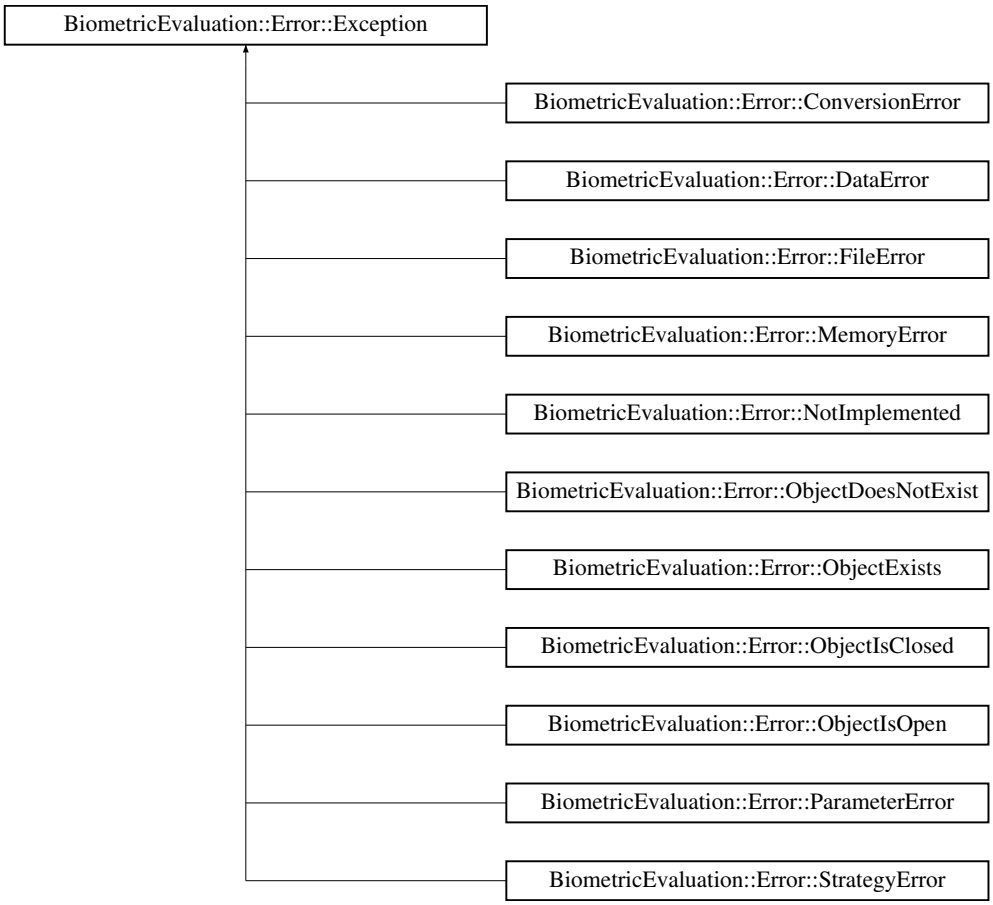
- be_feature_an2k7minutiae.h

E.31 BiometricEvaluation::Error::Exception Class Reference

The parent class of all BiometricEvaluation exceptions.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::Exception:



Public Member Functions

- Exception ()
- Exception (string info)
- string getInfo ()

E.31.1 Detailed Description

The parent class of all BiometricEvaluation exceptions. The classes derived from this class will have a default information string set indicating the type of exception. Any additional information string is appended to that string.

E.31.2 Constructor & Destructor Documentation

E.31.2.1 BiometricEvaluation::Error::Exception::Exception ( )

Construct an Exception object without an information string.

Returns

The Exception object.

E.31.2.2 BiometricEvaluation::Error::Exception::Exception (string *info*)

Construct an Exception object with an information string.

Parameters

`[in]` ***info*** The information string associated with the exception.

Returns

The Exception object.

E.31.3 Member Function Documentation

E.31.3.1 string BiometricEvaluation::Error::Exception::getInfo ( )

Obtain the information string associated with the exception.

Returns

The information string.

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

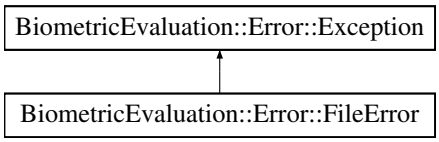
- be_error_exception.h

E.32 BiometricEvaluation::Error::FileError Class Reference

File error when opening, reading, writing, etc.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::FileError:



Public Member Functions

- FileError ()
- FileError (string info)

E.32.1 Detailed Description

File error when opening, reading, writing, etc.

E.32.2 Constructor & Destructor Documentation

E.32.2.1 BiometricEvaluation::Error::FileError::FileError ( )

Construct a FileError object with the default information string.

Returns

The FileError object.

E.32.2.2 BiometricEvaluation::Error::FileError::FileError (string *info*)

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

Returns

The FileError object.

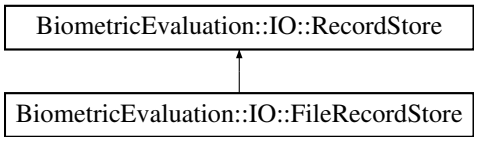
The documentation for this class was generated from the following file:

- be_error_exception.h

E.33 BiometricEvaluation::IO::FileRecordStore Class Reference

`#include <be_io_filerecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::FileRecordStore:



Public Member Functions

- FileRecordStore (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- FileRecordStore (const string &name, const string &parentDir, uint8_t mode=IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t getSpaceUsed () throw (Error::StrategyError)

  Obtain real storage utilization.

- void insert (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
- void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t read (const string &key, void *const data) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void replace (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t length (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void flush (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t sequence (string &key, void *const data, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)

- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)

Protected Member Functions

- string **canonicalName** (const string &name) const

E.33.1 Detailed Description

Class to represent the record store data storage mechanism implemented as files for each record.

Note

For the methods that take a key parameter, Error::StrategyError will be thrown if the key string is not compliant. A FileRecordStore has the additional requirement that a key name may not contain path delimiter characters ('/' and '\'), or begin with whitespace.

E.33.2 Constructor & Destructor Documentation

E.33.2.1 BiometricEvaluation::IO::FileRecordStore::FileRecordStore (const string & *name,* const string & *description,* const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new FileRecordStore, read/write mode.

Parameters

`[in]` ***name*** The name of the store.

`[in]` ***description*** The store's description.

`[in]` ***parentDir*** The directory where the store is to be created.

Exceptions

Error::ObjectExists The store already exists.

Error::StrategyError An error occurred when accessing the underlying file system.

E.33.2.2 BiometricEvaluation::IO::FileRecordStore::FileRecordStore (const string & *name*, const string & *parentDir*, uint8_t *mode* = `IO::READWRITE`) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing FileRecordStore.

Parameters

`[in]` ***name*** The name of the store.

`[in]` ***parentDir*** The directory where the store is to be created.

`[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.

E.33.3 Member Function Documentation

E.33.3.1 uint64_t BiometricEvaluation::IO::FileRecordStore::getSpaceUsed ( ) throw (Error::StrategyError) `[virtual]`

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.33.3.2 void BiometricEvaluation::IO::FileRecordStore::insert (const string & *key*, const void *const *data*, const uint64_t *size*) throw (Error::ObjectExists, Error::StrategyError) `[virtual]`

Insert a record into the store.

Parameters

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

[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.

E.33.3.3 void BiometricEvaluation::IO::FileRecordStore::remove (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Remove a record from the store.

Parameters

[in] ***key*** The key of the record to be removed.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.33.3.4 uint64_t BiometricEvaluation::IO::FileRecordStore::read (const string & *key*, void *const *data*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

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

Parameters

[in] ***key*** The key of the record to be read.

[in] ***data*** Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.33.3.5 virtual void BiometricEvaluation::IO::FileRecordStore::replace (const string & *key*, const void *const *data*, const uint64_t *size*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Replace a complete record in a store.

Parameters

[in] ***key*** The key of the record to be replaced.

[in] ***data*** The data for the record.

[in] ***size*** The size of data.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.33.3.6 virtual uint64_t BiometricEvaluation::IO::FileRecordStore::length (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Return the length of a record.

Parameters

[in] ***key*** The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.33.3.7 void BiometricEvaluation::IO::FileRecordStore::flush (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Commit the record's data to storage.

Parameters

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

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.33.3.8 uint64_t BiometricEvaluation::IO::FileRecordStore::sequence (string & *key*, void *const *data*, int *cursor* = BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Sequence through a RecordStore, returning the key/data pairs. Sequencing means to start at some point in the store and return the record, then repeatedly calling the sequencor to return the next record. The starting point is typically the the first record, and is set to that when the RecordStore object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

[out] ***key*** The key of the currently sequenced record.

[in] ***data*** Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.

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

Returns

The length of the record currently in sequence.

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.33.3.9 void BiometricEvaluation::IO::FileRecordStore::setCursorAtKey (string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Set the sequence cursor to an arbitrary position within the RecordStore, starting at key. Key will be the first record returned from the next call to sequence().

Parameters

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

Exceptions

Error::ObjectDoesNotExist A record for the key does not exist.

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

Implements BiometricEvaluation::IO::RecordStore.

E.33.3.10 void BiometricEvaluation::IO::FileRecordStore::changeName (const string & *name*) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Change the name of the RecordStore.

Parameters

[in] ***name*** The new name for the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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

- be_io_filerecstore.h

E.34 BiometricEvaluation::Finger::FingerImageCode Class Reference

`#include <be_finger.h>`

Public Types

- enum **Kind** {

  EJI = 0, **RolledTip**, **FullFingerRolled**, **FullFingerPlainLeft**,

  FullFingerPlainCenter, **FullFingerPlainRight**, **ProximalSegment**, **DistalSegment**,

  MedialSegment, **NA** }

E.34.1 Detailed Description

Joint and tip codes.

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

- be_finger.h

E.35 BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintRead Struct Reference

Representation of information about a fingerprint reader system.

`#include <be_feature_an2k7minutiae.h>`

Public Attributes

- string name
- EncodingMethod::Kind method
- string equipment

E.35.1 Detailed Description

Representation of information about a fingerprint reader system.

E.35.2 Member Data Documentation

E.35.2.1 string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::name

Name for system that encoded minutiae

E.35.2.2 EncodingMethod::Kind BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::method

Method used to encoded minutiae

E.35.2.3 string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::equipment

Optional ID for equipment used in system

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

- be_feature_an2k7minutiae.h

E.36 BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition Struct Reference

Locations of an individual finger segment in a slap.

`#include <be_finger_an2kview_capture.h>`

Public Member Functions

- FingerSegmentPosition (const Finger::Position::Kind fingerPosition, const Image::CoordinateSet coordinates)

  Create an FingerSegmentPosition struct.

Public Attributes

- Finger::Position::Kind fingerPosition
- Image::CoordinateSet coordinates

E.36.1 Detailed Description

Locations of an individual finger segment in a slap.

E.36.2 Constructor & Destructor Documentation

E.36.2.1 BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::FingerSegmentPosi (const Finger::Position::Kind *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.

E.36.3 Member Data Documentation

E.36.3.1 Finger::Position::Kind BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::fingerPosition

Finger depicted in this segment

E.36.3.2 Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::coordinates

Points composing the segmented polygon

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

- be_finger_an2kview_capture.h

E.37 BiometricEvaluation::Image::Image Class Reference

Represent attributes common to all images.

`#include <be_image_image.h>`

Inheritance diagram for BiometricEvaluation::Image::Image:

Public Member Functions

- Image (const uint8_t *data, const uint64_t size, const Size dimensions, const uint32_t depth, const Resolution resolution, const CompressionAlgorithm::Kind compression) throw (Error::DataError, Error::StrategyError)

  Parent constructor for all Image classes.

- Image (const uint8_t *data, const uint64_t size, const CompressionAlgorithm::Kind compression) throw (Error::DataError, Error::StrategyError)

  Parent constructor for all Image classes.

- CompressionAlgorithm::Kind getCompressionAlgorithm () const

  Accessor for the CompressionAlgorithm of the image.

- Resolution getResolution () const

  Accessor for the resolution of the image.

- Memory::AutoArray< uint8_t > getData () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.

- virtual Memory::AutoArray< uint8_t > getRawData () const =0 throw (Error::DataError)

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

- virtual Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const =0 throw (Error::DataError, Error::ParameterError)

  Accessor for decompressed data in grayscale.

- Size getDimensions () const

Accessor for the dimensions of the image in pixels.

- uint32_t getDepth () const

  Accessor for the color depth of the image in bits.

Static Public Member Functions

- static uint64_t valueInColorspace (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.

Static Public Attributes

- static const uint32_t bitsPerComponent = 8
- static const uint8_t max8BitColor = 255
- static const uint16_t max16BitColor = 65535
- static const uint32_t max24BitColor = 16777215
- static const uint32_t max32BitColor = 4294967295
- static const uint64_t max48BitColor = 281474976710655

Protected Member Functions

- void setResolution (const Resolution resolution)

  Mutator for the resolution of the image .

- void setDimensions (const Size dimensions)

  Mutator for the dimensions of the image in pixels.

- void setDepth (const uint32_t depth)

  Mutator for the color depth of the image in bits.

Protected Attributes

- Memory::AutoArray< uint8_t > _raw_data

E.37.1 Detailed Description

Represent attributes common to all images. Images are represented by their size, depth, and resolution on the X and Y axes. The image data can be of any format, raw, JPEG, etc. Implementations of this abstraction provide the getRawData() method to convert image data to 'raw' format.

Image resolution is in pixels per centimeter, and the coordinate system has the origin at the upper left of the image.

E.37.2 Constructor & Destructor Documentation

E.37.2.1 BiometricEvaluation::Image::Image::Image (const uint8_t * *data*, const uint64_t *size*, const Size *dimensions*, const uint32_t *depth*, const Resolution *resolution*, const CompressionAlgorithm::Kind *compression*) throw (Error::DataError, Error::StrategyError)

Parent constructor for all Image classes.

Parameters

[in] ***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.

E.37.2.2 BiometricEvaluation::Image::Image::Image (const uint8_t * *data*, const uint64_t *size*, const CompressionAlgorithm::Kind *compression*) throw (Error::DataError, Error::StrategyError)

Parent constructor for all Image classes.

Parameters

[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.

E.37.3 Member Function Documentation

E.37.3.1 CompressionAlgorithm::Kind BiometricEvaluation::Image::Image::getCompressionAlgorithm ( ) const

Accessor for the CompressionAlgorithm of the image.

Returns

Type of compression used on the data that will be returned from getData().

E.37.3.2 Resolution BiometricEvaluation::Image::Image::getResolution ( ) const

Accessor for the resolution of the image.

Returns

Resolution struct

E.37.3.3 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::Image::getData ( ) const

Accessor for the image data. The data returned is likely encoded in a specialized format.

Returns

Image data.

Reimplemented in BiometricEvaluation::Image::RawImage.

E.37.3.4 virtual Memory::AutoArray<uint8_t> BiometricEvaluation::Image::Image::getRawData ( ) const throw (Error::DataError) [pure virtual]

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

Returns

Raw image data.

Exceptions

Error::DataError Error decompressing image data.

Implemented in BiometricEvaluation::Image::JPEG, BiometricEvaluation::Image::JPEG2000, BiometricEvaluation::Image::JPEGL, BiometricEvaluation::Image::NetPBM, BiometricEvaluation::Image::PNG, BiometricEvaluation::Image::RawImage, and BiometricEvaluation::Image::WSQ.

E.37.3.5 virtual Memory::AutoArray<uint8_t> BiometricEvaluation::Image::Image::getRawGrayscaleData (uint8_t *depth* = `8`) const throw (Error::DataError, Error::ParameterError) [pure virtual]

Accessor for decompressed data in grayscale.

Parameters

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

Returns

Raw image buffer.

Exceptions

Error::DataError Error decompressing image data.

Error::ParameterError Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implemented in BiometricEvaluation::Image::JPEG, BiometricEvaluation::Image::JPEG2000, BiometricEvaluation::Image::JPEGL, BiometricEvaluation::Image::NetPBM, BiometricEvaluation::Image::PNG, BiometricEvaluation::Image::RawImage, and BiometricEvaluation::Image::WSQ.

E.37.3.6 Size BiometricEvaluation::Image::Image::getDimensions ( ) const

Accessor for the dimensions of the image in pixels.

Returns

Coordinate object containing dimensions in pixels.

E.37.3.7 uint32_t BiometricEvaluation::Image::Image::getDepth ( ) const

Accessor for the color depth of the image in bits.

Returns

The color depth of the image (bit).

E.37.3.8 static uint64_t BiometricEvaluation::Image::Image::valueInColorspace (uint64_t *color*, uint64_t *maxColorValue*, uint8_t *depth*) [static]

Calculate an equivalent color value for a color in an alternate colorspace.

Parameters

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.

E.37.3.9 void BiometricEvaluation::Image::Image::setResolution (const Resolution *resolution*) [protected]

Mutator for the resolution of the image .

Parameters

[in] ***resolution*** Resolution struct.

E.37.3.10 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).

E.37.3.11 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).

E.37.4 Member Data Documentation

E.37.4.1 const uint32_t BiometricEvaluation::Image::Image::bitsPerComponent = 8 [static]

Number of bits per color component

E.37.4.2 const uint8_t BiometricEvaluation::Image::Image::max8BitColor = 255 [static]

Maximum value for a color for 8-bit depth

E.37.4.3 const uint16_t BiometricEvaluation::Image::Image::max16BitColor = 65535 [static]

Maximum value for a color for 16-bit depth

E.37.4.4 const uint32_t BiometricEvaluation::Image::Image::max24BitColor = 16777215 [static]

Maximum value for a color in 24-bit depth

E.37.4.5 const uint32_t BiometricEvaluation::Image::Image::max32BitColor = 4294967295 [static]

Maximum value for a color in 32-bit depth

E.37.4.6 const uint64_t BiometricEvaluation::Image::Image::max48BitColor = 281474976710655 [static]

Maximum value for a color in 48-bit depth

E.37.4.7 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::Image::_raw_data [mutable, protected]

Raw image data, populated on demand

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

- be_image_image.h

E.38 BiometricEvaluation::Finger::Impression Class Reference

Finger and palm impression types.

`#include <be_finger.h>`

Public Types

- enum **Kind** {

  LiveScanPlain = 0, **LiveScanRolled**, **NonLiveScanPlain**, **NonLiveScanRolled**,

  LatentImpression, **LatentTracing**, **LatentPhoto**, **LatentLift**,

  LiveScanVerticalSwipe, **LiveScanPalm**, **NonLiveScanPalm**, **LatentPalmImpression**,

  LatentPalmTracing, **LatentPalmPhoto**, **LatentPalmLift**, **LiveScanOpticalContactPlain**,

  LiveScanOpticalContactRolled, **LiveScanNonOpticalContactPlain**, **LiveScanNonOpticalContactRolled**, **LiveScanOpticalContactlessPlain**,

  LiveScanOpticalContactlessRolled, **LiveScanNonOpticalContactlessPlain**, **LiveScanNonOpticalContactlessRolled**, **Other**,

Unknown }

E.38.1 Detailed Description

Finger and palm impression types.

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

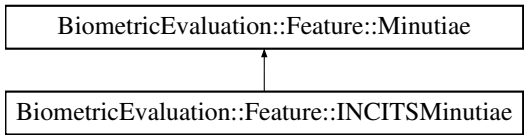
- be_finger.h

E.39 BiometricEvaluation::Feature::INCITSMinutiae Class Reference

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

`#include <be_feature_incitsminutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::INCITSMinutiae:



Public Member Functions

- MinutiaeFormat::Kind getFormat () const

  Obtain the minutiae format kind.

- MinutiaPointSet getMinutiaPoints () const

  Obtain the set of finger minutiae data points. The set may be empty.

- RidgeCountItemSet getRidgeCountItems () const

  Obtain the set of ridge count data items. The set may be empty.

- CorePointSet getCores () const

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

- DeltaPointSet getDeltas () const

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

- INCITSMinutiae (const MinutiaPointSet &mps, const RidgeCountItemSet &rcis, const CorePointSet &cps, const DeltaPointSet &dps)

  Construct an INCITS Minutiae object from its components.

- INCITSMinutiae ()

  Default constructor for an INCITS Minutiae object.

- void setMinutiaPoints (const MinutiaPointSet &mps)

  Mutator for the minutiae point set.

- void setRidgeCountItems (const RidgeCountItemSet &rcis)

  Mutator for the ridge count items.

- void setCorePointSet (const CorePointSet &cps)

  Mutator for the set of core points.

- void setDeltaPointSet (const DeltaPointSet &dps)

  Mutator for the set of delta points.

Static Public Attributes

- static const string **FMR_ANSI_SPEC_VERSION**
- static const string **FMR_ISO_SPEC_VERSION**
- static const string **FMR_ANSI07_SPEC_VERSION**
- static const uint8_t **FMR_SPEC_VERSION_LEN** = 4
- static const uint32_t **FED_HEADER_LENGTH** = 4
- static const uint32_t **FED_RCD_ITEM_LENGTH** = 3
- static const uint16_t **FMD_MINUTIA_TYPE_MASK** = 0xC000
- static const uint16_t **FMD_RESERVED_MASK** = 0xC000
- static const uint16_t **FMD_MINUTIA_TYPE_SHIFT** = 14
- static const uint16_t **FMD_RESERVED_SHIFT** = 14
- static const uint16_t **FMD_X_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_Y_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_MASK** = 0xC0
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_SHIFT** = 6
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_ANGLE_MASK** = 0x3F
- static const uint16_t **FMD_MIN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MAX_MINUTIA_QUALITY** = 100

- static const uint16_t **FMD_UNKNOWN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MIN_MINUTIA_ANGLE** = 0
- static const uint16_t **FMD_MAX_MINUTIA_ANGLE** = 179
- static const uint16_t **FMD_MAX_MINUTIA_ISONC_ANGLE** = 255
- static const uint16_t **FMD_MAX_MINUTIA_ISOCC_ANGLE** = 63
- static const uint16_t **FMD_ANSI_ANGLE_UNIT** = 2
- static const uint16_t **FMD_ISO_ANGLE_UNIT** = (360.0 / 256.0)
- static const uint16_t **FMD_ISOCC_ANGLE_UNIT** = (360.0 / 64.0)
- static const uint16_t **FMD_MINUTIA_TYPE_OTHER** = 0
- static const uint16_t **FMD_MINUTIA_TYPE_RIDGE_ENDING** = 1
- static const uint16_t **FMD_MINUTIA_TYPE_BIFURCATION** = 2
- static const uint16_t **FMR_MIN_FINGER_QUALITY** = 0
- static const uint16_t **FMR_MAX_FINGER_QUALITY** = 100
- static const uint16_t **ISO_UNKNOWN_FINGER_QUALITY** = 0
- static const uint16_t **FED_RESERVED** = 0x0000
- static const uint16_t **FED_RIDGE_COUNT** = 0x0001
- static const uint16_t **FED_CORE_AND_DELTA** = 0x0002
- static const uint16_t **RCE_NONSPECIFIC** = 0x00
- static const uint16_t **RCE_FOUR_NEIGHBOR** = 0x01
- static const uint16_t **RCE_EIGHT_NEIGHBOR** = 0x02
- static const uint16_t **CORE_TYPE_NONANGULAR** = 0x00
- static const uint16_t **CORE_TYPE_ANGULAR** = 0x01
- static const uint16_t **DELTA_TYPE_NONANGULAR** = 0x00
- static const uint16_t **DELTA_TYPE_ANGULAR** = 0x01

E.39.1 Detailed Description

A class to represent a set of minutiae in an ANSI/INCITS record. The base INCITSMinutiae class is responsible for reading minutiae data points and extended data. Each minutiae point, ridge count item, core, and delta is represented in the native ANSI/INCITS format. Objects of this base class cannot be instantiated, but rather derived classes are used to represent minutiae data taken from the INCITS-derived record formats.

E.39.2 Constructor & Destructor Documentation

E.39.2.1 BiometricEvaluation::Feature::INCITSMinutiae::INCITSMinutiae (const MinutiaPointSet & *mps,* const RidgeCountItemSet & *rcis,* const CorePointSet & *cps,* const DeltaPointSet & *dps*)

Construct an INCITS Minutiae object from its components.

The buffer index must be set to the location in the buffer to start reading minutiae data points and extended data.

Parameters

[in] ***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.

E.39.3 Member Function Documentation

E.39.3.1 void BiometricEvaluation::Feature::INCITSMinutiae::setMinutiaPoints (const MinutiaPointSet & *mps*)

Mutator for the minutiae point set.

Parameters

mps] The minutiae points.

E.39.3.2 void BiometricEvaluation::Feature::INCITSMinutiae::setRidgeCountItems (const RidgeCountItemSet & *rcis*)

Mutator for the ridge count items.

Parameters

[in] ***rcis*** The set of ridge count items.

E.39.3.3 void BiometricEvaluation::Feature::INCITSMinutiae::setCorePointSet (const CorePointSet & *cps*)

Mutator for the set of core points.

Parameters

[in] ***cps*** The set of core points.

E.39.3.4 void BiometricEvaluation::Feature::INCITSMinutiae::setDeltaPointSet (const DeltaPointSet & *dps*)

Mutator for the set of delta points.

Parameters

`[in]` ***dps*** The set of delta point items.

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

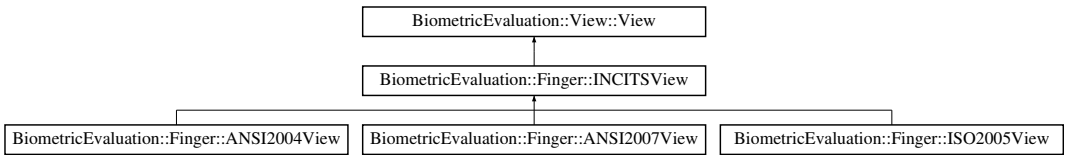
- be_feature_incitsminutiae.h

E.40 BiometricEvaluation::Finger::INCITSView Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_incitsview.h>`

Inheritance diagram for BiometricEvaluation::Finger::INCITSView:



Public Member Functions

- Feature::INCITSMinutiae getMinutiaeData () const

  Obtain the set of minutiae records.

- Finger::Position::Kind getPosition () const

  Obtain the finger position.

- Finger::Impression::Kind getImpressionType () const

  Obtain the finger impression code.

- uint32_t getQuality () const

  Obtain the finger quality value.

- uint16_t getCaptureEquipmentID () const

  Obtain the capture equipment identifier.

- bool isAppendixFCompliant () const

  Obtain the capture equipment compliance indicator for 'Appendix F'.

- tr1::shared_ptr< Image::Image > getImage () const

  Obtain the image used for the finger view.

- Image::Size getImageSize () const

  Obtain the image size.

- Image::Resolution getImageResolution () const

  Obtain the image resolution.

- uint32_t getImageDepth () const

  Obtain the image depth.

- Image::CompressionAlgorithm::Kind getCompressionAlgorithm () const

  Obtain the compression algorithm used on the image.

- Image::Resolution getScanResolution () const

  Obtain the image scan resolution.

Static Public Member Functions

- static Finger::Position::Kind convertPosition (int incitsFGP) throw (Error::DataError)

  Convert a finger postion code from an INCITS finger record to the common code.

- static Finger::Impression::Kind convertImpression (int incitsIMP) throw (Error::DataError)

  Convert a impression type code from an INCITS finger record to the common code.

Static Public Attributes

- static const uint32_t **FMR_ANSI2004_STANDARD** = 1
- static const uint32_t **FMR_ISO2005_STANDARD** = 2
- static const uint32_t **FMR_ANSI2007_STANDARD** = 3

- static const string **FMR_BASE_FORMAT_ID**
- static const uint32_t **FMR_SPEC_VERSION_LEN** = 4
- static const string **FMR_BASE_SPEC_VERSION**
- static const string **FMR_ANSI2007_SPEC_VERSION**
- static const uint16_t **FMR_HDR_SCANNER_ID_MASK** = 0x0FFF
- static const uint16_t **FMR_HDR_COMPLIANCE_MASK** = 0xF000
- static const uint8_t **FMR_HDR_COMPLIANCE_SHIFT** = 12
- static const uint16_t **FMR_HDR_APPENDIX_F_MASK** = 0x0008
- static const uint8_t **FVMR_VIEW_NUMBER_MASK** = 0xF0
- static const uint8_t **FVMR_VIEW_NUMBER_SHIFT** = 4
- static const uint8_t **FVMR_IMPRESSION_MASK** = 0x0F

Protected Member Functions

- INCITSView (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber) throw (Error::DataError, Error::FileError)

  Construct the common components of an INCITS finger view from records contained in files.

- INCITSView (const Memory::uint8Array &fmrBuffer, const Memory::uint8Array &firBuffer, const uint32_t viewNumber) throw (Error::DataError)

  Construct an INCITS finger view from records contained in buffers.

- Memory::uint8Array const & getFMRData () const

  Obtain a reference to the finger minutiae record data buffer.

- Memory::uint8Array const & getFIRData () const

  Obtain a reference to the finger image record data buffer.

- void setMinutiaeData (const Feature::INCITSMinutiae &fmd)

  Mutator for the Feature::INCITSMinutiae item.

- void setPosition (const Finger::Position::Kind &position)

  Mutator for the position.

- void setImpressionType (const Finger::Impression::Kind &impression)

  Mutator for the impression type.

- void setQuality (uint32_t quality)

  Mutator for the finger quality value.

- void setViewNumber (uint32_t viewNumber)

  Mutator for the finger view number.

- void setCaptureEquipmentID (uint16_t id)

  Mutator for the equipment ID.

- void setCBEFFProductIDs (uint16_t owner, uint16_t type)

  Mutator for the CBEFF Product ID owner and type.

- void setAppendixFCompliance (bool flag)

  Mutator for the Appendix F compliance indicator.

- void setImageSize (const Image::Size &imageSize)

  Mutator for the image size.

- void setImageResolution (const Image::Resolution &imageResolution)

  Mutator for the image resolution.

- void setScanResolution (const Image::Resolution &scanResolution)

  Mutator for the image scan resolution.

- void setImageData (const Memory::uint8Array &imageData)

  Mutator for the image data.

- void readFMRHeader (Memory::IndexedBuffer &buf, const uint32_t formatStandard) throw (Error::ParameterError, Error::DataError)

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

- void readFVMR (Memory::IndexedBuffer &buf) throw (Error::DataError)

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

- virtual Feature::MinutiaPointSet readMinutiaeDataPoints (Memory::IndexedBuffer &buf, uint32_t count) throw (Error::DataError)

  Read the minutiae data points, and extended data blocks.

- virtual void readExtendedDataBlock (Memory::IndexedBuffer &buf) throw (Error::DataError)

  Read the common extended data block.

- virtual Feature::RidgeCountItemSet readRidgeCountData (Memory::IndexedBuffer &buf, uint32_t dataLength) throw (Error::DataError)

  Read the ridge count data.

- virtual void readCoreDeltaData (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)=0 throw (Error::DataError)

  Read the core points data.

E.40.1 Detailed Description

A class to represent single finger view and derived information. A base Finger::INCITSView object represents an INCITS/ANSI or ISO finger view. This class defines the common interface for all ANSI/ISO views as well as common implementations. Subclasses specialize this class in order to represent other versions of the ANSI/ISO specs. Objects of this class cannot be created.

E.40.2 Constructor & Destructor Documentation

E.40.2.1 BiometricEvaluation::Finger::INCITSView::INCITSView (const std::string & *fmrFilename,* const std::string & *firFilename,* const uint32_t *viewNumber*) throw (Error::DataError, Error::FileError) [protected]

Construct the common components of an INCITS finger view from records contained in files.

See documentation in child classes of INCITS for information on constructing INCITS-derived finger views.

Parameters

[in] ***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.

E.40.2.2 BiometricEvaluation::Finger::INCITSView::INCITSView (const Memory::uint8Array & *fmrBuffer,* const Memory::uint8Array & *firBuffer,* const uint32_t *viewNumber*) throw (Error::DataError) [protected]

Construct an INCITS finger view from records contained in buffers.

See documentation in child classes of INCITS for information on constructing INCITS-derived finger views.

Parameters

[in] ***fmrFilename*** The buffer containing the complete finger minutiae record.

[in] ***firFilename*** The buffer containing the complete finger image record.

[in] ***viewNumber*** The finger view number to use.

E.40.3 Member Function Documentation

E.40.3.1 static Finger::Position::Kind BiometricEvaluation::Finger::INCITSView::convertPosition (int *incitsFGP*) throw (Error::DataError) [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.

E.40.3.2 static Finger::Impression::Kind BiometricEvaluation::Finger::INCITSView::convertImpression (int *incitsIMP*) throw (Error::DataError) [static]

Convert a impression type code from an INCITS finger record to the common code.

Parameters

[in] ***incitsFGP*** 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.

E.40.3.3 Finger::Position::Kind BiometricEvaluation::Finger::INCITSView::getPosition ( ) const

Obtain the finger position.

Returns

The finger position.

E.40.3.4 Finger::Impression::Kind BiometricEvaluation::Finger::INCITSView::getImpressionType ( ) const

Obtain the finger impression code.

Returns

The finger impression code.

E.40.3.5 uint32_t BiometricEvaluation::Finger::INCITSView::getQuality ( ) const

Obtain the finger quality value.

Returns

The finger quality value.

E.40.3.6 uint16_t BiometricEvaluation::Finger::INCITSView::getCaptureEquipmentID ( ) const

Obtain the capture equipment identifier.

Returns

The equipment ID.

E.40.3.7 bool BiometricEvaluation::Finger::INCITSView::isAppendixFCompliant ( ) const

Obtain the capture equipment compliance indicator for 'Appendix F'.

Returns

True if 'Appendix F' compliant, false otherwise.

E.40.3.8 tr1::shared_ptr<Image::Image> BiometricEvaluation::Finger::INCITSView::getImage ( ) const [virtual]

Obtain the image used for the finger view.

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

Implements BiometricEvaluation::View::View.

E.40.3.9 Image::Size BiometricEvaluation::Finger::INCITSView::getImageSize ( ) const [virtual]

Obtain the image size.

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

Implements BiometricEvaluation::View::View.

E.40.3.10 Image::Resolution BiometricEvaluation::Finger::INCITSView::getImageResolution ( ) const [virtual]

Obtain the image resolution.

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

Implements BiometricEvaluation::View::View.

E.40.3.11 uint32_t BiometricEvaluation::Finger::INCITSView::getImageDepth ( ) const [virtual]

Obtain the image depth.

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

Implements BiometricEvaluation::View::View.

E.40.3.12 Image::CompressionAlgorithm::Kind BiometricEvaluation::Finger::INCITSView::getCompressionAlgorithm ( ) const [virtual]

Obtain the compression algorithm used on the image.

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

Implements BiometricEvaluation::View::View.

E.40.3.13 Image::Resolution BiometricEvaluation::Finger::INCITSView::getScanResolution ( ) const [virtual]

Obtain the image scan resolution.

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

Implements BiometricEvaluation::View::View.

E.40.3.14 Memory::uint8Array const& BiometricEvaluation::Finger::INCITSView::getFMRData ( ) const [protected]

Obtain a reference to the finger minutiae record data buffer.

Returns

The entire finger minutiae record data.

E.40.3.15 Memory::uint8Array const& BiometricEvaluation::Finger::INCITSView::getFIRData ( ) const [protected]

Obtain a reference to the finger image record data buffer.

Returns

The entire finger image record data.

E.40.3.16 void BiometricEvaluation::Finger::INCITSView::setMinutiaeData (const Feature::INCITSMinutiae & *fmd*) [protected]

Mutator for the Feature::INCITSMinutiae item.

Parameters

[in] ***fmd*** The minutiae data object.

E.40.3.17 void BiometricEvaluation::Finger::INCITSView::setPosition (const Finger::Position::Kind & *position*) [protected]

Mutator for the position.

Parameters

[in] ***position*** The finger position.

E.40.3.18 void BiometricEvaluation::Finger::INCITSView::setImpressionType (const Finger::Impression::Kind & *impression*) [protected]

Mutator for the impression type.

Parameters

[in] ***impression*** The finger impression type code.

E.40.3.19 void BiometricEvaluation::Finger::INCITSView::setQuality (uint32_t *quality*) [protected]

Mutator for the finger quality value.

Parameters

[in] ***quality*** The quality value.

E.40.3.20 void BiometricEvaluation::Finger::INCITSView::setViewNumber (uint32_t *viewNumber*) [protected]

Mutator for the finger view number.

Parameters

[in] ***viewNumber*** The view number value.

E.40.3.21 void BiometricEvaluation::Finger::INCITSView::setCaptureEquipmentID (uint16_t *id*) [protected]

Mutator for the equipment ID.

Parameters

[in] ***id*** The equipment ID value.

E.40.3.22 void BiometricEvaluation::Finger::INCITSView::setCBEFFProductIDs (uint16_t *owner*, uint16_t *type*) [protected]

Mutator for the CBEFF Product ID owner and type.

Parameters

[in] ***Owner*** The CBEFF ID of the product owner.

[in] ***Type*** The CBEFF ID of the product type.

E.40.3.23 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.

E.40.3.24 void BiometricEvaluation::Finger::INCITSView::setImageSize (const Image::Size & *imageSize*) [protected]

Mutator for the image size.

Parameters

[in] ***imageSize*** The image size object.

E.40.3.25 void BiometricEvaluation::Finger::INCITSView::setImageResolution (const Image::Resolution & *imageResolution*) [protected]

Mutator for the image resolution.

Parameters

[in] ***imageResolution*** The image resolution object.

E.40.3.26 void BiometricEvaluation::Finger::INCITSView::setScanResolution (const Image::Resolution & *scanResolution*) [protected]

Mutator for the image scan resolution.

Parameters

[in] ***scanResolution*** The image scan resolution object.

E.40.3.27 void BiometricEvaluation::Finger::INCITSView::setImageData (const Memory::uint8Array & *imageData*) [protected]

Mutator for the image data.

Parameters

[in] ***imageData*** The image data object.

E.40.3.28 void BiometricEvaluation::Finger::INCITSView::readFMRHeader (Memory::IndexedBuffer & *buf,* const uint32_t *formatStandard*) throw (Error::ParameterError, Error::DataError) [protected]

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

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

Parameters

`[in]` ***buf*** The indexed buffer containing the record data. 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 FMR_ANSI2004_STANDARD or FMR_ISO2005_STANDARD.

Exceptions

ParameterError The specVersion parameter is incorrect.

DataError The INCITS record has invalid or missing data.

Reimplemented in BiometricEvaluation::Finger::ANSI2007View.

E.40.3.29 void BiometricEvaluation::Finger::INCITSView::readFVMR (Memory::IndexedBuffer & *buf*) throw (Error::DataError) [protected]

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

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

Parameters

`[in,out]` ***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.

Reimplemented in BiometricEvaluation::Finger::ANSI2007View.

E.40.3.30 virtual Feature::MinutiaPointSet BiometricEvaluation::Finger::INCITSView::readMinutiaeDataPoints (Memory::IndexedBuffer & *buf,* uint32_t *count*) throw (Error::DataError) [protected, virtual]

Read the minutiae data points, and extended data blocks.

Function to be implemented by derived classes to read the minutiae data points and extended data block according to the 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.

Exceptions

DataError The INCITS record has invalid or missing data.

E.40.3.31 virtual void BiometricEvaluation::Finger::INCITSView::readExtendedDataBlock (Memory::IndexedBuffer & *buf*) throw (Error::DataError) [protected, virtual]

Read the common extended data block.

Parameters

[in,out] ***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.

E.40.3.32 virtual Feature::RidgeCountItemSet BiometricEvaluation::Finger::INCITSView::readRidgeCountData (Memory::IndexedBuffer & *buf,* uint32_t *dataLength*) throw (Error::DataError) [protected, virtual]

Read the ridge count data.

This method reads data in the base INCITS format as defined in INCITS/ANSI 378-2004. This method may be overridden by derived classes to read data in a different record format.

Parameters

[in,out] ***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.

E.40.3.33 virtual void BiometricEvaluation::Finger::INCITSView::readCoreDeltaData (Memory::IndexedBuffer & *buf*, uint32_t *dataLength*, Feature::CorePointSet & *cores*, Feature::DeltaPointSet & *deltas*) throw (Error::DataError) [protected, pure virtual]

Read the core points data.

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

Parameters

[in,out] ***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::ANSI2004View, BiometricEvaluation::Finger::ANSI2007View, and BiometricEvaluation::Finger::ISO2005View.

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

- be_finger_incitsview.h

E.41 BiometricEvaluation::Memory::IndexedBuffer Class Reference

Manage a memory buffer with an index.

`#include <be_memory_indexedbuffer.h>`

Public Member Functions

- **operator uint8_t** * ()
- uint8_t * **operator->** ()
- IndexedBuffer & **operator=** (const IndexedBuffer &other)
- IndexedBuffer ()

  Create an indexed buffer of xero length.

- IndexedBuffer (uint32_t size)

  Create an indexed buffer of a given length.

- IndexedBuffer (uint8_t *data, uint32_t size)

  Create an indexed buffer around an existing buffer of a given length.

- IndexedBuffer (const IndexedBuffer &copy)

  Copy constructor.

- uint32_t getSize ()

  Obtain the current size of the buffer.

- uint32_t getIndex ()

  Obtain the current index into the buffer.

- void setIndex (uint32_t index) throw (Error::ParameterError)

  Set the current index into the buffer.

- uint8_t scanU8Val () throw (Error::DataError)

  Obtain the next element of the buffer and increment the current index value.

- uint16_t scanU16Val () throw (Error::DataError)

  Obtain the next two elements of the buffer and increment the current index value.

- uint16_t scanBeU16Val () throw (Error::DataError)

  Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.

- uint32_t scanU32Val () throw (Error::DataError)

  Obtain the next four elements of the buffer and increment the current index value by four.

- uint32_t scanBeU32Val () throw (Error::DataError)

  Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.

- uint64_t scanU64Val () throw (Error::DataError)

  Obtain the next eight elements of the buffer and increment the current index value by eight.

- uint32_t scan (void *buf, const uint32_t len) throw (Error::DataError)

  Obtain the next 'n' elements of the buffer and increment the current index value by n.

- uint8_t & operator[ ] (ptrdiff_t i)

  Subscripting operator.

- const uint8_t & operator[ ] (ptrdiff_t i) const

  Constant subscripting operator.

E.41.1 Detailed Description

Manage a memory buffer with an index. The memory buffer is treated as an array of unsigned eight bit values. This class provides safe access to the array with methods to retrieve 8/16/32/64-bit elements, or and arbitrary segment starting at the index, from the array while advancing the current index. An exception is thrown by these methods whenever the retrieval would reach beyond the size of the buffer.

The buffer can also be accessed directly by subscripting.

E.41.2 Constructor & Destructor Documentation

E.41.2.1 BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (uint8_t * *data*, uint32_t *size*)

Create an indexed buffer around an existing buffer of a given length.

An object constructed in this manner will not free the underlying data buffer.

E.41.3 Member Function Documentation

E.41.3.1 uint32_t BiometricEvaluation::Memory::IndexedBuffer::getSize ( )

Obtain the current size of the buffer.

Returns

The current buffer size.

E.41.3.2 uint32_t BiometricEvaluation::Memory::IndexedBuffer::getIndex ( )

Obtain the current index into the buffer.

Returns

The current buffer index.

E.41.3.3 void BiometricEvaluation::Memory::IndexedBuffer::setIndex (uint32_t *index*) throw (Error::ParameterError)

Set the current index into the buffer.

Parameters

`[in]` ***index*** The index value to set.

Exceptions

Error::ParameterError The index parameter is too large.

E.41.3.4 uint8_t BiometricEvaluation::Memory::IndexedBuffer::scanU8Val ( ) throw (Error::DataError)

Obtain the next element of the buffer and increment the current index value.

Exceptions

Error::DataError The buffer is exhausted.

Returns

The next element of the buffer as an unsigned 8-bit value.

E.41.3.5 uint16_t BiometricEvaluation::Memory::IndexedBuffer::scanU16Val ( ) throw (Error::DataError)

Obtain the next two elements of the buffer and increment the current index value.

Exceptions

Error::DataError The buffer is exhausted.

Returns

The next element of the buffer as an unsigned 16-bit value.

E.41.3.6 uint16_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU16Val ( ) throw (Error::DataError)

Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.

Exceptions

Error::DataError The buffer is exhausted.

Returns

The next element of the buffer as an unsigned 16-bit value.

E.41.3.7 uint32_t BiometricEvaluation::Memory::IndexedBuffer::scanU32Val ( ) throw (Error::DataError)

Obtain the next four elements of the buffer and increment the current index value by four.

Exceptions

Error::DataError The buffer is exhausted.

Returns

The next element of the buffer as an unsigned 32-bit value.

E.41.3.8 uint32_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU32Val ( ) throw (Error::DataError)

Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.

Exceptions

Error::DataError The buffer is exhausted.

Returns

The next element of the buffer as an unsigned 32-bit value.

E.41.3.9 uint64_t BiometricEvaluation::Memory::IndexedBuffer::scanU64Val ( ) throw (Error::DataError)

Obtain the next eight elements of the buffer and increment the current index value by eight.

Exceptions

Error::DataError The buffer is exhausted.

Returns

The next element of the buffer as an unsigned 64-bit value.

E.41.3.10 uint32_t BiometricEvaluation::Memory::IndexedBuffer::scan (void * *buf*, const uint32_t *len*) throw (Error::DataError)

Obtain the next 'n' elements of the buffer and increment the current index value by n.

Parameters

`[in]` ***buf*** Buffer to store the copied data.

size The number of elements to copy.

Exceptions

Error::DataError The buffer is exhausted.

Returns

The number of elements copied.

E.41.3.11 uint8_t& BiometricEvaluation::Memory::IndexedBuffer::operator[ ] (ptrdiff_t *i*)

Subscripting operator.

Provides array-like access to elements of the buffer. This operation will not affect the current index value.

Parameters

`[in]` ***i*** The subscript.

Returns

Reference to element 'i' of the buffer.

E.41.3.12 const uint8_t& BiometricEvaluation::Memory::IndexedBuffer::operator[ ] (ptrdiff_t *i*) const

Constant subscripting operator.

Provides read-only array-like access to elements of the buffer. This operation will not affect the current index value.

Parameters

[in] ***i*** The subscript.

Returns

Reference to const element 'i' of the buffer.

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

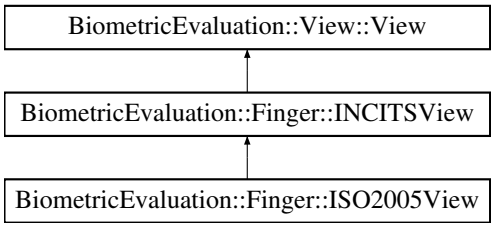
- be_memory_indexedbuffer.h

E.42 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) throw (Error::DataError, Error::FileError)

  Construct an ISO-2005 finger view from records contained in files.

- ISO2005View (const Memory::uint8Array &fmrBuffer, const Memory::uint8Array &firBuffer, const uint32_t viewNumber) throw (Error::DataError)

  Construct an ISO-2005 finger view from records contained in buffers.

Static Public Attributes

- static const uint16_t **CORE_TYPE_MASK** = 0xC000
- static const uint16_t **CORE_TYPE_SHIFT** = 14
- static const uint16_t **CORE_NUM_CORES_MASK** = 0x3F
- static const uint16_t **CORE_X_COORD_MASK** = 0x3FFF
- static const uint16_t **CORE_Y_COORD_MASK** = 0x3FFF
- static const uint16_t **CORE_MIN_NUM** = 0
- static const uint16_t **DELTA_TYPE_MASK** = 0xC000
- static const uint16_t **DELTA_TYPE_SHIFT** = 14
- static const uint16_t **DELTA_NUM_DELTAS_MASK** = 0x3F
- static const uint16_t **DELTA_X_COORD_MASK** = 0x3FFF
- static const uint16_t **DELTA_Y_COORD_MASK** = 0x3FFF

Protected Member Functions

- virtual void readCoreDeltaData (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas) throw (Error::DataError)

  Read the core points data.

E.42.1 Detailed Description

A class to represent single finger view and derived information. A Finger::ISO2005View object represents a finger view from a ISO/IEC-2005 Finger Minutiae Record.

E.42.2 Constructor & Destructor Documentation

E.42.2.1 BiometricEvaluation::Finger::ISO2005View::ISO2005View (const std::string & *fmrFilename,* const std::string & *firFilename,* const uint32_t *viewNumber*) throw (Error::DataError, Error::FileError)

Construct an ISO-2005 finger view from records contained in files.

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

Parameters

[in] ***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.

E.42.2.2 BiometricEvaluation::Finger::ISO2005View::ISO2005View (const Memory::uint8Array & *fmrBuffer,* const Memory::uint8Array & *firBuffer,* const uint32_t *viewNumber*) throw (Error::DataError)

Construct an ISO-2005 finger view from records contained in buffers.

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

Parameters

[in] ***fmrFilename*** The buffer containing the complete finger minutiae record.

[in] ***firFilename*** The buffer containing the complete finger image record.

[in] ***viewNumber*** The finger view number to use.

E.42.3 Member Function Documentation

E.42.3.1 virtual void BiometricEvaluation::Finger::ISO2005View::readCoreDeltaData (Memory::IndexedBuffer & *buf,* uint32_t *dataLength,* Feature::CorePointSet & *cores,* Feature::DeltaPointSet & *deltas*) throw (Error::DataError) [protected, virtual]

Read the core points data.

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

Parameters

[in,out] ***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.

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

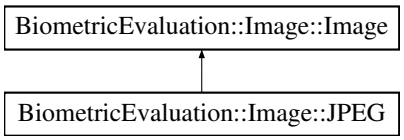
- be_finger_iso2005view.h

E.43 BiometricEvaluation::Image::JPEG Class Reference

A JPEG-encoded image.

`#include <be_image_jpeg.h>`

Inheritance diagram for BiometricEvaluation::Image::JPEG:



Public Member Functions

- **JPEG** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)

  Accessor for decompressed data in grayscale.

- Memory::AutoArray< uint8_t > getRawData () const throw (Error::DataError)

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

Static Public Member Functions

- static bool isJPEG (const uint8_t *data, const size_t size)

E.43.1 Detailed Description

A JPEG-encoded image.

E.43.2 Member Function Documentation

E.43.2.1 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::JPEG::getRawGrayscaleData (uint8_t *depth* = 8) const throw (Error::DataError, Error::ParameterError) [virtual]

Accessor for decompressed data in grayscale.

Parameters

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

Returns

Raw image buffer.

Exceptions

Error::DataError Error decompressing image data.

Error::ParameterError Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

E.43.2.2 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::JPEG::getRawData ( ) const throw (Error::DataError) [virtual]

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

Returns

Raw image data.

Exceptions

Error::DataError Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

E.43.2.3 static bool BiometricEvaluation::Image::JPEG::isJPEG (const uint8_t * *data*, const size_t *size*) [static]

Whether or not data is a Lossy JPEG image.

Parameters

[in] ***data*** The buffer to check.

[in] ***size*** The size of data.

Returns

true if data appears to be a Lossy JPEG image, false otherwise

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

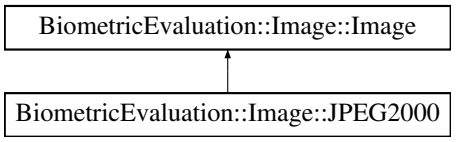
- be_image_jpeg.h

E.44 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 OPJ_CODEC_FORMAT codec=CODEC_JP2) throw (Error::DataError, Error::StrategyError)

Create a new JPEG2000 object.

- Memory::AutoArray< uint8_t > getRawData () const throw (Error::DataError)

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

- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)

  Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isJPEG2000 (const uint8_t *data)

E.44.1 Detailed Description

A JPEG-2000-encoded image.

E.44.2 Constructor & Destructor Documentation

E.44.2.1 BiometricEvaluation::Image::JPEG2000::JPEG2000 (const uint8_t * *data*, const uint64_t *size*, const OPJ_CODEC_FORMAT *codec* = `CODEC_JP2`) throw (Error::DataError, Error::StrategyError)

Create a new JPEG2000 object.

Parameters

`[in]` ***data*** The image data.

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

`[in]` ***codec*** The codec used to encode data.

Exceptions

Error::DataError Error manipulating data.

Error::StrategyError Error while creating Image.

E.44.3 Member Function Documentation

E.44.3.1 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::JPEG2000::getRawData ( ) const throw (Error::DataError) [virtual]

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

Returns

Raw image data.

Exceptions

Error::DataError Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

E.44.3.2 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::JPEG2000::getRawGrayscaleData (uint8_t *depth* = *8*) const throw (Error::DataError, Error::ParameterError) [virtual]

Accessor for decompressed data in grayscale.

Parameters

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

Returns

Raw image buffer.

Exceptions

Error::DataError Error decompressing image data.

Error::ParameterError Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

E.44.3.3 static bool BiometricEvaluation::Image::JPEG2000::isJPEG2000 (const uint8_t * *data*) [static]

Whether or not data is a JPEG-2000 image.

Parameters

[in] ***data*** The buffer to check.

Returns

true if data appears to be a JPEG-2000 image, false otherwise.

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

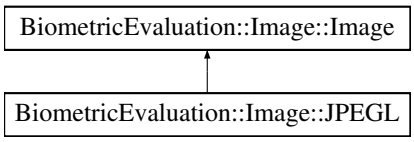
- be_image_jpeg2000.h

E.45 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) throw (Error::DataError, Error::StrategyError)
- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)

  Accessor for decompressed data in grayscale.

- Memory::AutoArray< uint8_t > getRawData () const throw (Error::DataError)

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

Static Public Member Functions

- static bool isJPEGL (const uint8_t *data, const size_t size)

E.45.1 Detailed Description

A Lossless JPEG-encoded image.

E.45.2 Member Function Documentation

E.45.2.1 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::JPEGL::getRawGrayscaleData (uint8_t *depth* = `8`) const throw (Error::DataError, Error::ParameterError) `[virtual]`

Accessor for decompressed data in grayscale.

Parameters

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

Returns

Raw image buffer.

Exceptions

Error::DataError Error decompressing image data.

Error::ParameterError Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

E.45.2.2 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::JPEGL::getRawData ( ) const throw (Error::DataError) [virtual]

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

Returns

Raw image data.

Exceptions

Error::DataError Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

E.45.2.3 static bool BiometricEvaluation::Image::JPEGL::isJPEGL (const uint8_t * *data*, const size_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.

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

- be_image_jpegl.h

E.46 BiometricEvaluation::IO::LogCabinet Class Reference

`#include <be_io_logcabinet.h>`

Public Member Functions

- LogCabinet (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)

- LogCabinet (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- tr1::shared_ptr< LogSheet > newLogSheet (const string &name, const string &description) throw (Error::ObjectExists, Error::StrategyError)
- string getName ()
- string getDescription ()
- unsigned int getCount ()

Static Public Member Functions

- static void remove (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)

E.46.1 Detailed Description

A class to represent a collection of log sheets.

E.46.2 Constructor & Destructor Documentation

E.46.2.1 BiometricEvaluation::IO::LogCabinet::LogCabinet (const string & *name,* const string & *description,* const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new LogCabinet in the file system.

Parameters

`[in]` ***name*** The name of the LogCabinet to be created.

`[in]` ***description*** The text used to describe the cabinet.

`[in]` ***parentDir*** Where, in the file system, the cabinet is to be stored. This directory must exist.

Returns

An object representing the new log cabinet.

Exceptions

Error::ObjectExists The cabinet was previously created.

Error::StrategyError

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

E.46.2.2 BiometricEvaluation::IO::LogCabinet::LogCabinet (const string & *name,* const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing LogCabinet.

Parameters

[in] ***name*** The name of the LogCabinet to be created.

[in] ***description*** The text used to describe the cabinet.

[in] ***parentDir*** Where, in the file system, the cabinet is to be stored. This directory must exist.

Returns

An object representing the log cabinet.

Exceptions

Error::ObjectDoesNotExist The cabinet does not exist in the file system.

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

E.46.3 Member Function Documentation

E.46.3.1 tr1::shared_ptr<LogSheet> BiometricEvaluation::IO::LogCabinet::newLogSheet (const string & *name,* const string & *description*) throw (Error::ObjectExists, Error::StrategyError)

Create a new LogSheet within the LogCabinet.

Parameters

[in] ***name*** The name of the LogSheet to be created.

[in] ***description*** The text used to describe the sheet. This text is written into the log file prior to any entries.

[in] ***parentDir*** Where, in the file system, the sheet is to be stored. This directory must exist.

Returns

An object pointer to the new log sheet.

Exceptions

Error::ObjectExists The sheet was previously created.

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

E.46.3.2 string BiometricEvaluation::IO::LogCabinet::getName ( )

Obtain the name of the LogCabinet.

@ returns The name of the LogCabinet.

E.46.3.3 string BiometricEvaluation::IO::LogCabinet::getDescription ( )

Obtain the description of the LogCabinet.

@ returns The description of the LogCabinet.

E.46.3.4 unsigned int BiometricEvaluation::IO::LogCabinet::getCount ( )

Obtain the number of items in the LogCabinet.

@ returns The number of LogSheets manages by the cabinet.

E.46.3.5 static void BiometricEvaluation::IO::LogCabinet::remove (const string & *name*, const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[static]`

Remove a LogCabinet.

Parameters

`[in]` ***name*** The name of the LogCabinet to be removed.

`[in]` ***parentDir*** Where, in the file system, the sheet is to be stored. This directory must exist.

Exceptions

Error::ObjectDoesNotExist The LogCabinet does not exist.

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

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

- be_io_logcabinet.h

E.47 BiometricEvaluation::IO::LogSheet Class Reference

A class to represent a single logging mechanism.

`#include <be_io_logcabinet.h>`

Public Member Functions

- LogSheet (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)

  Create a new log sheet.

- LogSheet (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Open an existing new log sheet for appending.

- void write (const string &entry) throw (Error::StrategyError)
- void writeComment (const string &comment) throw (Error::StrategyError)
- void newEntry () throw (Error::StrategyError)
- string getCurrentEntry ()
- void resetCurrentEntry ()
- uint32_t getCurrentEntryNumber ()
- void sync () throw (Error::StrategyError)
- void setAutoSync (bool state)

Static Public Attributes

- static const char CommentDelimiter = '#'
- static const char EntryDelimiter = 'E'
- static const string DescriptionTag

E.47.1 Detailed Description

A class to represent a single logging mechanism. A LogSheet is a string stream, so applications can write into the stream as a staging area using the << operator, then start a new entry by calling newEntry(). Entries in the log file are prefixed with an entry number, which is incremented when the entry is written (either by directly calling write(), or calling newEntry()).

A LogSheet object can be constructed and passed back to the client by the LogCabinet object. All sheets created in the manner are placed in a common area maintained by the cabinet.

Note

By default, the entries in the LogSheet may not be immediately written to the file system, depending on the buffering behavior of the operating system. Applications can force a write by invoking sync(), or force a write at every new log entry by invoking setAutoSync(true).
Entries created by applications may be composed of more than one line (each separated by the newline character). The text at the beginning of a line should not "look like" an entry number:
Edddd
i.e. the entry delimiter followed by some digits. LogSheet won't check for that condition, but any existing LogSheet that is re-opened for append may have an incorrect starting entry number.

E.47.2 Constructor & Destructor Documentation

E.47.2.1 BiometricEvaluation::IO::LogSheet::LogSheet (const string & *name,* const string & *description,* const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new log sheet.

Parameters

`[in]` ***name*** The name of the LogSheet to be created.

`[in]` ***description*** The text used to describe the sheet. This text is written into the log file prior to any entries.

`[in]` ***parentDir*** Where, in the file system, the sheet is to be stored. This directory must exist.

Returns

An object representing the new log sheet.

Exceptions

Error::ObjectExists The sheet was previously created.

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

E.47.2.2 BiometricEvaluation::IO::LogSheet::LogSheet (const string & *name,* const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing new log sheet for appending.

On open, the current entry counter is set to the last entry number plus one.

Note

Opening a large LogSheet may be a costly operation.

Parameters

[in] ***name*** The name of the LogSheet to be opened.

[in] ***parentDir*** Where, in the file system, the sheet is stored.

Returns

An object representing the existing log sheet.

Exceptions

Error::ObjectDoesNotExist The sheet does not exist.

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

E.47.3 Member Function Documentation

E.47.3.1 void BiometricEvaluation::IO::LogSheet::write (const string & *entry*) throw (Error::StrategyError)

Write a string as an entry to the log file. This does not affect the current log entry buffer, but does increment the entry number.

Parameters

[in] ***entry*** The text of the log entry.

Exceptions

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

E.47.3.2 void BiometricEvaluation::IO::LogSheet::writeComment (const string & *comment*) throw (Error::StrategyError)

Write a string as a comment to the log file. This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

[in] ***comment*** The text of the comment.

Exceptions

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

E.47.3.3 void BiometricEvaluation::IO::LogSheet::newEntry ( ) throw (Error::StrategyError)

Start a new entry, causing the existing entry to be closed. Applications do not have to call this method for the first entry, however, as the stream is ready for writing upon construction.

Exceptions

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

E.47.3.4 string BiometricEvaluation::IO::LogSheet::getCurrentEntry ( )

Obtain the contents of the current entry currently under construction.

Returns

The text of the current entry.

E.47.3.5 void BiometricEvaluation::IO::LogSheet::resetCurrentEntry ( )

Reset the current entry buffer to the beginning.

E.47.3.6 uint32_t BiometricEvaluation::IO::LogSheet::getCurrentEntryNumber ( )

Obtain the current entry number.

Returns

The current entry number.

E.47.3.7 void BiometricEvaluation::IO::LogSheet::sync ( ) throw (Error::StrategyError)

Synchronize any buffered data to the underlying log file. This syncing is dependent on the behavior of the underlying filesystem and operating system.

Exceptions

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

E.47.3.8 void BiometricEvaluation::IO::LogSheet::setAutoSync (bool *state*)

Turn on/off auto-sync of the data. Applications can gain loggin performance by turning off auto-sysnc, 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.

E.47.4 Member Data Documentation

E.47.4.1 const char BiometricEvaluation::IO::LogSheet::CommentDelimiter = '#' [static]

The delimiter for a comment line in the log sheet.

E.47.4.2 const char BiometricEvaluation::IO::LogSheet::EntryDelimiter = 'E' [static]

The delimiter for an entry line in the log sheet.

E.47.4.3 const string BiometricEvaluation::IO::LogSheet::DescriptionTag [static]

The tag for the description string.

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

- be_io_logcabinet.h

E.48 BiometricEvaluation::IO::ManifestEntry Struct Reference

`#include <be_io_archiverecstore.h>`

Public Attributes

- long offset
- uint64_t size

E.48.1 Detailed Description

Info about a single archive element

E.48.2 Member Data Documentation

E.48.2.1 long BiometricEvaluation::IO::ManifestEntry::offset

The offset from the beginning of the file/memory

E.48.2.2 uint64_t BiometricEvaluation::IO::ManifestEntry::size

The length from offset this element spans

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

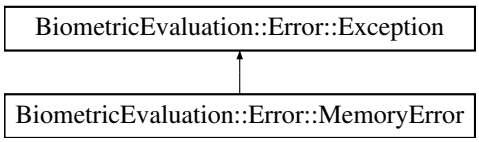
- be_io_archiverecstore.h

E.49 BiometricEvaluation::Error::MemoryError Class Reference

An error occurred when allocating an object.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::MemoryError:



Public Member Functions

- MemoryError ()
- MemoryError (string info)

E.49.1 Detailed Description

An error occurred when allocating an object.

E.49.2 Constructor & Destructor Documentation

E.49.2.1 BiometricEvaluation::Error::MemoryError::MemoryError ( )

Construct a MemoryError object with the default information string.

Returns

The MemoryError object.

E.49.2.2 BiometricEvaluation::Error::MemoryError::MemoryError (string *info*)

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

Returns

The MemoryError object.

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

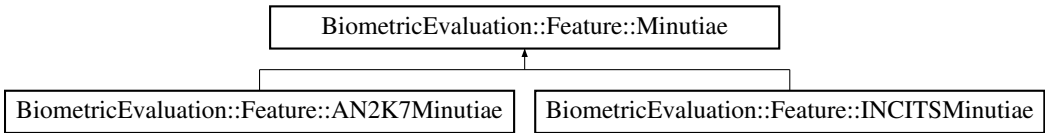
- be_error_exception.h

E.50 BiometricEvaluation::Feature::Minutiae Class Reference

A class to represent a set of minutiae data points.

`#include <be_feature_minutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::Minutiae:



Public Member Functions

- virtual MinutiaeFormat::Kind getFormat () const =0

  Obtain the minutiae format kind.

- virtual MinutiaPointSet getMinutiaPoints () const =0

  Obtain the set of finger minutiae data points. The set may be empty.

- virtual RidgeCountItemSet getRidgeCountItems () const =0

  Obtain the set of ridge count data items. The set may be empty.

- virtual CorePointSet getCores () const =0

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

- virtual DeltaPointSet getDeltas () const =0

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

E.50.1 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.

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

- be_feature_minutiae.h

E.51 BiometricEvaluation::Feature::MinutiaeFormat Class Reference

Enumerate the minutiae format standards.

`#include <be_feature_minutiae.h>`

Public Types

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

E.51.1 Detailed Description

Enumerate the minutiae format standards.

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

- be_feature_minutiae.h

E.52 BiometricEvaluation::Feature::MinutiaeType Class Reference

Enumerate the types of minutiae: Ridge Ending, Bifurcation, Compound, or other.

`#include <be_feature_minutiae.h>`

Public Types

- enum **Kind** { **RidgeEnding** = 0, **Bifurcation**, **Compound**, **Other** }

E.52.1 Detailed Description

Enumerate the types of minutiae: Ridge Ending, Bifurcation, Compound, or other.

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

- be_feature_minutiae.h

E.53 BiometricEvaluation::Feature::MinutiaPoint Struct Reference

Representation of a finger minutiae data point.

`#include <be_feature_minutiae.h>`

Public Attributes

- unsigned int **index**
- bool **has_type**
- MinutiaeType::Kind **type**
- Image::Coordinate **coordinate**
- unsigned int **theta**

- bool **has_quality**
- unsigned int **quality**

E.53.1 Detailed Description

Representation of a finger minutiae data point.

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

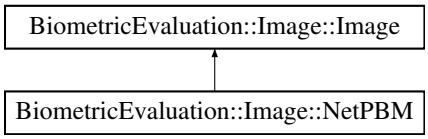
- be_feature_minutiae.h

E.54 BiometricEvaluation::Image::NetPBM Class Reference

A NetPBM-encoded image.

`#include <be_image_netpbm.h>`

Inheritance diagram for BiometricEvaluation::Image::NetPBM:



Public Types

- enum **Kind** {

  ASCIIPortableBitmap = 1, **ASCIIPortableGraymap** = 2, **ASCIIPortablePixmap** = 3, **BinaryPortableBitmap** = 4,

  BinaryPortableGraymap = 5, **BinaryPortablePixmap** = 6 }

Public Member Functions

- **NetPBM** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- Memory::AutoArray< uint8_t > getRawData () const throw (Error::DataError)

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

- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)

  Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isNetPBM (const uint8_t *data, const size_t size)
- static void skipLine (Memory::uint8Array &data, size_t &offset) throw (out_of_range)

  Skip an entire line of input, placing offset at the first character after the newline.

- static void skipComment (Memory::uint8Array &data, size_t &offset) throw (out_of_range)

  Skip a block of comments in input.

- static string getNextValue (Memory::uint8Array &data, size_t &offset, size_t sizeOfValue=0)

  Obtain the next space-separated value from data, beginning at offset.

- static Memory::uint8Array ASCIIBitmapTo8Bit (Memory::uint8Array &bitmap, uint32_t width, uint32_t height) throw (out_of_range)

  Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.

- static Memory::uint8Array ASCIIPixmapToBinaryPixmap (Memory::uint8Array &ASCIIBuf, uint32_t width, uint32_t height, uint8_t depth, uint32_t maxColor) throw (out_of_range, Error::ParameterError)

  Convert an ASCII pixel map buffer into a binary pixel map buffer.

- static Memory::uint8Array BinaryBitmapTo8Bit (Memory::uint8Array &bitmap, uint32_t width, uint32_t height) throw (out_of_range)

  Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

E.54.1 Detailed Description

A NetPBM-encoded image.

Note

While a NetPBM file can contain more than one image, this class will only support the first image found in any file, also known as the "plain" NetPBM format.

E.54.2 Member Function Documentation

E.54.2.1 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::NetPBM::getRawData ( ) const throw (Error::DataError) [virtual]

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

Returns

Raw image data.

Exceptions

Error::DataError Error decompressing image data.

Note

The raw data returned from this method is encoded at the same bit depth as the compressed data, except in the case of 1-bit (bitmap) images, which are expanded to 8-bit.

Implements BiometricEvaluation::Image::Image.

E.54.2.2 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::NetPBM::getRawGrayscaleData (uint8_t *depth* = *8*) const throw (Error::DataError, Error::ParameterError) [virtual]

Accessor for decompressed data in grayscale.

Parameters

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

Returns

Raw image buffer.

Exceptions

Error::DataError Error decompressing image data.

Error::ParameterError Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

E.54.2.3 static bool BiometricEvaluation::Image::NetPBM::isNetPBM (const uint8_t * *data*, const size_t *size*) [static]

Whether or not data is a netpbm image.

Parameters

[in] ***data*** The buffer to check.

[in] ***size*** The size of data.

Returns

true if data appears to be a netpbm image, false otherwise.

E.54.2.4 static void BiometricEvaluation::Image::NetPBM::skipLine (Memory::uint8Array & *data*, size_t & *offset*) throw (out_of_range) [static]

Skip an entire line of input, placing offset at the first character after the newline.

Parameters

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.

E.54.2.5 static void BiometricEvaluation::Image::NetPBM::skipComment (Memory::uint8Array & *data*, size_t & *offset*) throw (out_of_range) [static]

Skip a block of comments in input.

Parameters

offset Position within data from which the rest of the line should be read.

sizeOfValue If non-zero, the expected size of the next value. Useful for payload that does not need to be white-space separated.

Exceptions

out_of_range End of line not encountered before end of data or on last line of data.

E.54.2.6 static string BiometricEvaluation::Image::NetPBM::getNextValue (Memory::uint8Array & *data,* size_t & *offset,* size_t *sizeOfValue = 0*) [static]

Obtain the next space-separated value from data, beginning at offset.

Parameters

data Buffer where next value will be obtained.

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.

E.54.2.7 static Memory::uint8Array BiometricEvaluation::Image::NetPBM::ASCIIBitmapTo8Bit (Memory::uint8Array & *bitmap,* uint32_t *width,* uint32_t *height*) throw (out_of_range) [static]

Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Parameters

bitmap Bitmap data buffer.

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.

E.54.2.8 static Memory::uint8Array BiometricEvaluation::Image::NetPBM::ASCIIPixmapToBinaryPixmap (Memory::uint8Array & *ASCIIBuf,* uint32_t *width,* uint32_t *height,* uint8_t *depth,* uint32_t *maxColor*) throw (out_of_range, Error::ParameterError) `[static]`

Convert an ASCII pixel map buffer into a binary pixel map buffer.

Parameters

ASCIIBuf ASCII pixel map data buffer.

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.

E.54.2.9 static Memory::uint8Array BiometricEvaluation::Image::NetPBM::BinaryBitmapTo8Bit (Memory::uint8Array & *bitmap,* uint32_t *width,* uint32_t *height*) throw (out_of_range) `[static]`

Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Parameters

bitmap Bitmap data buffer.

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.

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

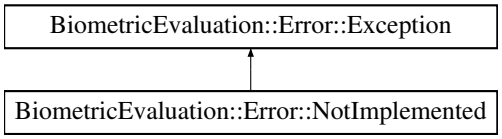
- be_image_netpbm.h

E.55 BiometricEvaluation::Error::NotImplemented Class Reference

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

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::NotImplemented:



Public Member Functions

- NotImplemented ()
- NotImplemented (string info)

E.55.1 Detailed Description

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

E.55.2 Constructor & Destructor Documentation

E.55.2.1 BiometricEvaluation::Error::NotImplemented::NotImplemented ( )

Construct a NotImplemented object with the default information string.

Returns

The NotImplemented object.

E.55.2.2 BiometricEvaluation::Error::NotImplemented::NotImplemented (string *info*)

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

Returns

The NotImplemented object.

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

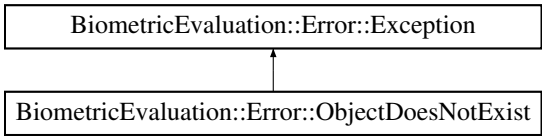
- be_error_exception.h

E.56 BiometricEvaluation::Error::ObjectDoesNotExist Class Reference

The named object does not exist.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectDoesNotExist:



Public Member Functions

- ObjectDoesNotExist ()
- ObjectDoesNotExist (string info)

E.56.1 Detailed Description

The named object does not exist.

E.56.2 Constructor & Destructor Documentation

E.56.2.1 BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( )

Construct a ObjectDoesNotExist object with the default information string.

Returns

The ObjectDoesNotExist object.

E.56.2.2 BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist (**string** *info*)

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

Returns

The ObjectDoesNotExist object.

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

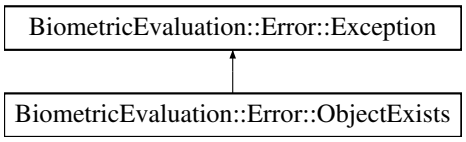
- be_error_exception.h

E.57 BiometricEvaluation::Error::ObjectExists Class Reference

The named object exists and will not be replaced.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectExists:



Public Member Functions

- ObjectExists ()
- ObjectExists (string info)

E.57.1 Detailed Description

The named object exists and will not be replaced.

E.57.2 Constructor & Destructor Documentation

E.57.2.1 BiometricEvaluation::Error::ObjectExists::ObjectExists ( )

Construct a ObjectExists object with the default information string.

Returns

The ObjectExists object.

E.57.2.2 BiometricEvaluation::Error::ObjectExists::ObjectExists (string *info*)

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

Returns

The ObjectExists object.

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

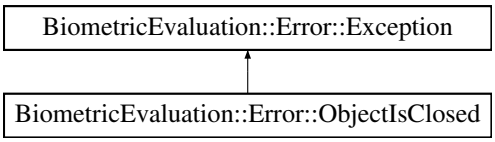
- be_error_exception.h

E.58 BiometricEvaluation::Error::ObjectIsClosed Class Reference

The object is closed.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectIsClosed:



Public Member Functions

- ObjectIsClosed ()
- ObjectIsClosed (string info)

E.58.1 Detailed Description

The object is closed.

E.58.2 Constructor & Destructor Documentation

E.58.2.1 BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ( )

Construct a ObjectIsClosed object with the default information string.

Returns

The ObjectIsClosed object.

E.58.2.2 BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed (string *info*)

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

Returns

The ObjectIsClosed object.

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

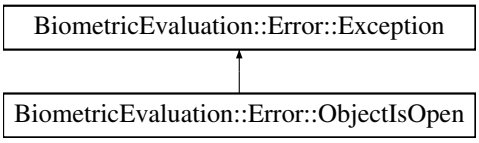
- be_error_exception.h

E.59 BiometricEvaluation::Error::ObjectIsOpen Class Reference

The object is already opened.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectIsOpen:



Public Member Functions

- ObjectIsOpen ()
- ObjectIsOpen (string info)

E.59.1 Detailed Description

The object is already opened.

E.59.2 Constructor & Destructor Documentation

E.59.2.1 BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ( )

Construct a ObjectIsOpen object with the default information string.

Returns

The ObjectIsOpen object.

E.59.2.2 BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen (string *info*)

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

Returns

The ObjectIsOpen object.

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

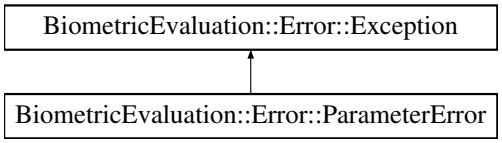
- be_error_exception.h

E.60 BiometricEvaluation::Error::ParameterError Class Reference

An invalid parameter was passed to a constructor or method.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ParameterError:



Public Member Functions

- ParameterError ()
- ParameterError (string info)

E.60.1 Detailed Description

An invalid parameter was passed to a constructor or method.

E.60.2 Constructor & Destructor Documentation

E.60.2.1 BiometricEvaluation::Error::ParameterError::ParameterError ( )

Construct a ParameterError object with the default information string.

Returns

The ParameterError object.

E.60.2.2 BiometricEvaluation::Error::ParameterError::ParameterError (string *info*)

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

Returns

The ParameterError object.

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

- be_error_exception.h

E.61 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification Class Reference

Pattern classification codes.

`#include <be_feature_an2k7minutiae.h>`

Classes

- struct Entry

Public Types

- enum **Kind** {
  PlainArch = 0, **TentedArch**, **RadialLoop**, **UlnarLoop**,
  PlainWhorl, **CentralPocketLoop**, **DoubleLoop**, **AccidentalWhorl**,
  Whorl, **RightSlantLoop**, **LeftSlantLoop**, **Scar**,
  Amputation, **Unknown** }
- typedef struct Entry **Entry**

E.61.1 Detailed Description

Pattern classification codes.

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

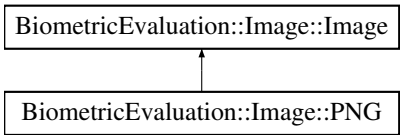
- be_feature_an2k7minutiae.h

E.62 BiometricEvaluation::Image::PNG Class Reference

A PNG-encoded image.

`#include <be_image_png.h>`

Inheritance diagram for BiometricEvaluation::Image::PNG:



Classes

- struct **png_buffer**

  Wrapper for reading PNG-encoded data from an AutoArray with libpng.

Public Member Functions

- **PNG** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- Memory::AutoArray< uint8_t > getRawData () const throw (Error::DataError)

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

- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)

  Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isPNG (const uint8_t *data)

E.62.1 Detailed Description

A PNG-encoded image.

E.62.2 Member Function Documentation

E.62.2.1 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::PNG::getRawData ( ) const throw (Error::DataError) [virtual]

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

Returns

Raw image data.

Exceptions

Error::DataError Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

E.62.2.2 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::PNG::getRawGrayscaleData (uint8_t *depth* = 8) const throw (Error::DataError, Error::ParameterError) [virtual]

Accessor for decompressed data in grayscale.

Parameters

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

Returns

Raw image buffer.

Exceptions

Error::DataError Error decompressing image data.

Error::ParameterError Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

E.62.2.3 static bool BiometricEvaluation::Image::PNG::isPNG (const uint8_t * *data*) [static]

Whether or not data is a PNG image.

Parameters

[in] ***data*** The buffer to check.

Returns

true if data appears to be a PNG image, false otherwise

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

- be_image_png.h

E.63 BiometricEvaluation::Finger::Position Class Reference

Finger position codes.

`#include <be_finger.h>`

Public Types

- enum **Kind** {
  Unknown = 0, **RightThumb** = 1, **RightIndex** = 2, **RightMiddle** = 3,
  RightRing = 4, **RightLittle** = 5, **LeftThumb** = 6, **LeftIndex** = 7,
  LeftMiddle = 8, **LeftRing** = 9, **LeftLittle** = 10, **PlainRightThumb** = 11,
  PlainLeftThumb = 12, **PlainRightFourFingers** = 13, **PlainLeftFourFingers** = 14, **LeftRightThumbs** = 15,
  EJI = 19 }

E.63.1 Detailed Description

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

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

- be_finger.h

E.64 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::Kind &fingerView, FingerImageCode::Kind &segment, Image::CoordinateSet &coordinates)

  Construct a PrintPositionCoordinate.

Public Attributes

- FingerImageCode::Kind fingerView
- FingerImageCode::Kind segment
- Image::CoordinateSet coordinates

E.64.1 Detailed Description

Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

E.64.2 Constructor & Destructor Documentation

E.64.2.1 BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::PrintPositionCoord (FingerImageCode::Kind & *fingerView,* FingerImageCode::Kind & *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).

E.64.3 Member Data Documentation

E.64.3.1 FingerImageCode::Kind BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::fingerView

Full finger view being bounded

E.64.3.2 FingerImageCode::Kind BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::segment

Segment within full finger view bound

E.64.3.3 Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::coordinates

Two coordinates forming bounding box

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

- be_finger_an2kview_varres.h

E.65 BiometricEvaluation::IO::Properties Class Reference

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

`#include <be_io_properties.h>`

Public Types

- typedef PropertiesMap::const_iterator **Properties_iter**

Public Member Functions

- Properties (const string &filename, uint8_t mode=IO::READWRITE) throw (Error::StrategyError, Error::FileError)
- void setProperty (const string &property, const string &value) throw (Error::StrategyError)
- void setPropertyFromInteger (const string &property, int64_t value) throw (Error::StrategyError)

- void removeProperty (const string &property) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- string getProperty (const string &property) throw (Error::ObjectDoesNotExist)
- int64_t getPropertyAsInteger (const string &property) throw (Error::ObjectDoesNotExist, Error::ConversionError)
- void sync () throw (Error::FileError, Error::StrategyError)
- void changeName (const string &filename) throw (Error::StrategyError)

E.65.1 Detailed Description

A Properties class is used to maintain key/value pairs of strings, with each property matched to one value. The properties are read from a file that is specified in the constructor, and will be created if it does not exist.

An example file might look like this:

```
*     Name = John Smith
*     Age = 32
*     Favorite Hex Number = 0xffff
*
```

For property keys and values, leading and trailing whitespace is removed, therefore a the call

```
props->setProperty("  My property   ", "   A Value  ");
```

results in an entry in the property file as

```
*     My property = A value
*
```

Therefore, the property names "Foo", " Foo", "Foo " are equivalent.

E.65.2 Constructor & Destructor Documentation

E.65.2.1 BiometricEvaluation::IO::Properties::Properties (const string & *filename*, uint8_t *mode* = *IO::READWRITE*) throw (Error::StrategyError, Error::FileError)

Construct a new Properties object from an existing or to be created properties file. The constructor will create the file when it does not exist.

Parameters

[in] ***filename*** The name of the file to store the properties. This can be the empty string, meaning the properties are to be stored in memory only.

[in] ***mode*** The read/write mode of the object.

Returns

An object representing the properties set.

Exceptions

Error::StrategyError A line in the properties file is malformed.

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

E.65.3 Member Function Documentation

E.65.3.1 void BiometricEvaluation::IO::Properties::setProperty (const string & *property*, const string & *value*) throw (Error::StrategyError)

Set a property with a value. Both the property and value will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise, the property will be created.

Parameters

[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.

E.65.3.2 void BiometricEvaluation::IO::Properties::setPropertyFromInteger (const string & *property*, int64_t *value*) throw (Error::StrategyError)

Set a property with an integer value. The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise the property will be created.

Parameters

[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.

E.65.3.3 void BiometricEvaluation::IO::Properties::removeProperty (const string & *property*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

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.

E.65.3.4 string BiometricEvaluation::IO::Properties::getProperty (const string & *property*) throw (Error::ObjectDoesNotExist)

Retrieve a property value as a string object.

Parameters

`[in]` ***property*** The name of the property to get.

Exceptions

Error::ObjectDoesNotExist The named property does not exist.

E.65.3.5 int64_t BiometricEvaluation::IO::Properties::getPropertyAsInteger (const string & *property*) throw (Error::ObjectDoesNotExist, Error::ConversionError)

Retrieve a property value as an integer value. Integer value strings for properties can represent either decimal or hexadecimal values, which must be preceded with either "0x" or "0X".

Parameters

`[in]` ***property*** The name of the property to get.

Exceptions

Error::ObjectDoesNotExist The named property does not exist.

Error::ConversionError The property value cannot be converted, usually due to non-numeric characters in the string.

E.65.3.6 void BiometricEvaluation::IO::Properties::sync ( ) throw (Error::FileError, Error::StrategyError)

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

Exceptions

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

Error::StrategyError The object was constructed with NULL as the file name, or is read-only.

E.65.3.7 void BiometricEvaluation::IO::Properties::changeName (const string & *filename*) throw (Error::StrategyError)

Change the name of the Properties, which means changing the name of the underlying file that stores the properties. The empty string ("") can be used to indicate no backing file.

Note

No check is made that the file is writeable at this time.

Parameters

`[in]` ***filename*** The name of the properties file.

Exceptions

Error::StrategyError The object is read-only.

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

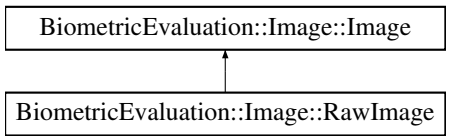
- be_io_properties.h

E.66 BiometricEvaluation::Image::RawImage Class Reference

An image with no encoding or compression.

`#include <be_image_rawimage.h>`

Inheritance diagram for BiometricEvaluation::Image::RawImage:



Public Member Functions

- **RawImage** (const uint8_t *data, const uint64_t size, const Size dimensions, const unsigned int depth, const Resolution resolution)
- Memory::AutoArray< uint8_t > getData () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.

- Memory::AutoArray< uint8_t > getRawData () const throw (Error::DataError)

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

- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)

  Accessor for decompressed data in grayscale.

E.66.1 Detailed Description

An image with no encoding or compression.

E.66.2 Member Function Documentation

E.66.2.1 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::RawImage::getData ( ) const

Accessor for the image data. The data returned is likely encoded in a specialized format.

Returns

Image data.

Reimplemented from BiometricEvaluation::Image::Image.

E.66.2.2 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::RawImage::getRawData ( ) const throw (Error::DataError) [virtual]

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

Returns

Raw image data.

Exceptions

Error::DataError Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

E.66.2.3 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::RawImage::getRawGrayscaleData (uint8_t *depth* = *8*) const throw (Error::DataError, Error::ParameterError) [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

Raw image buffer.

Exceptions

Error::DataError Error decompressing image data.

Error::ParameterError Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

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

- be_image_rawimage.h

E.67 BiometricEvaluation::IO::RecordStore Class Reference

A class to represent a data storage mechanism.

`#include <be_io_recordstore.h>`

Inheritance diagram for BiometricEvaluation::IO::RecordStore:

Public Member Functions

- RecordStore (const string &name, const string &description, const string &type, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- RecordStore (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- string getName () const
- string getDescription () const
- unsigned int getCount () const
- virtual void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- virtual void changeDescription (const string &description) throw (Error::StrategyError)
- virtual uint64_t getSpaceUsed () throw (Error::StrategyError)

  Obtain real storage utilization.

- virtual void sync () throw (Error::StrategyError)
- virtual void insert (const string &key, const void *const data, const uint64_t size)=0 throw (Error::ObjectExists, Error::StrategyError)
- virtual void remove (const string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t read (const string &key, void *const data)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void replace (const string &key, const void *const data, const uint64_t size)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t length (const string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

- virtual void flush (const string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t sequence (string &key, void *const data=NULL, int cursor=BE_RECSTORE_SEQ_NEXT)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void setCursorAtKey (string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

Static Public Member Functions

- static tr1::shared_ptr< RecordStore > openRecordStore (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Open an existing RecordStore and return a managed pointer to the the object representing that store.

- static tr1::shared_ptr< RecordStore > createRecordStore (const string &name, const string &description, const string &type, const string &destDir) throw (Error::ObjectExists, Error::StrategyError)

  Create a new RecordStore and return a managed pointer to the the object representing that store.

- static void removeRecordStore (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- static void mergeRecordStores (const string &mergedName, const string &mergedDescription, const string &parentDir, const string &type, RecordStore *recordStores[ ], size_t numRecordStores) throw (Error::ObjectExists, Error::StrategyError)
- static void mergeRecordStores (const string &mergedName, const string &mergedDescription, const string &parentDir, const string &type, tr1::shared_ptr< RecordStore > recordStores[ ], size_t numRecordStores) throw (Error::ObjectExists, Error::StrategyError)

Static Public Attributes

- static const string CONTROLFILENAME
- static const string NAMEPROPERTY
- static const string DESCRIPTIONPROPERTY
- static const string COUNTPROPERTY
- static const string TYPEPROPERTY
- static const string BERKELEYDBTYPE
- static const string ARCHIVETYPE
- static const string FILETYPE

- static const int BE_RECSTORE_SEQ_START = 1
- static const int BE_RECSTORE_SEQ_NEXT = 2

Protected Member Functions

- uint8_t **getMode** () const
- string **getDirectory** () const
- string **getParentDirectory** () const
- string **canonicalName** (const string &name) const
- int **getCursor** () const
- void **setCursor** (int cursor)

E.67.1 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.

See also

IO::ArchiveRecordStore, IO::DBRecordStore, IO::FileRecordStore.

E.67.2 Constructor & Destructor Documentation

E.67.2.1 BiometricEvaluation::IO::RecordStore::RecordStore (const string & *name*, const string & *description*, const string & *type*, const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Constructor to create a new RecordStore.

Parameters

[in] ***name*** The name of the RecordStore to be created.

[in] ***description*** The text used to describe the store.

[in] ***type*** The type of RecordStore.

[in] ***parentDir*** Where, in the file system, the store is to be rooted. This directory must exist.

Returns

An object representing the new, empty store.

Exceptions

Error::ObjectExists The store was previously created, or the directory where it would be created exists.

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

E.67.2.2 BiometricEvaluation::IO::RecordStore::RecordStore (const string & *name,* const string & *parentDir,* uint8_t *mode* = ***READWRITE***) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Constructor to open an existing RecordStore.

Parameters

`[in]` ***name*** The name of the store to be opened.

`[in]` ***parentDir*** Where, in the file system, the store is rooted.

`[in]` ***mode*** The type of access a client of this RecordStore has.

Returns

An object representing the existing store.

Exceptions

Error::ObjectDoesNotExist The RecordStore does not exist.

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

E.67.3 Member Function Documentation

E.67.3.1 string BiometricEvaluation::IO::RecordStore::getName ( ) const

Return the name of the RecordStore.

Returns

The RecordStore's name.

E.67.3.2 string BiometricEvaluation::IO::RecordStore::getDescription ( ) const

Obtain a textual description of the RecordStore.

Returns

The RecordStore's description.

E.67.3.3 unsigned int BiometricEvaluation::IO::RecordStore::getCount ( ) const

Obtain the number of items in the RecordStore.

Returns

The number of items in the RecordStore.

E.67.3.4 virtual void BiometricEvaluation::IO::RecordStore::changeName (const string & *name*) throw (Error::ObjectExists, Error::StrategyError) `[virtual]`

Change the name of the RecordStore.

Parameters

`[in]` ***name*** The new name for the RecordStore.

Exceptions

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

Reimplemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.5 virtual void BiometricEvaluation::IO::RecordStore::changeDescription (const string & *description*) throw (Error::StrategyError) `[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.

E.67.3.6 virtual uint64_t BiometricEvaluation::IO::RecordStore::getSpaceUsed ( ) throw (Error::StrategyError) [virtual]

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.7 virtual void BiometricEvaluation::IO::RecordStore::sync ( ) throw (Error::StrategyError) [virtual]

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented in BiometricEvaluation::IO::ArchiveRecordStore, and BiometricEvaluation::IO::DBRecordStore.

E.67.3.8 virtual void BiometricEvaluation::IO::RecordStore::insert (const string & *key*, const void *const *data*, const uint64_t *size*) throw (Error::ObjectExists, Error::StrategyError) [pure virtual]

Insert a record into the store.

Parameters

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

[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::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.9 virtual void BiometricEvaluation::IO::RecordStore::remove (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

Remove a record from the store.

Parameters

[in] ***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::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.10 virtual uint64_t BiometricEvaluation::IO::RecordStore::read (const string & *key,* void *const *const *data*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

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

Parameters

[in] ***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.

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.11 virtual void BiometricEvaluation::IO::RecordStore::replace (const string & *key*, const void *const *data*, const uint64_t *size*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

Replace a complete record in a store.

Parameters

[in] ***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::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.12 virtual uint64_t BiometricEvaluation::IO::RecordStore::length (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [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::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.13 virtual void BiometricEvaluation::IO::RecordStore::flush (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [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::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.14 virtual uint64_t BiometricEvaluation::IO::RecordStore::sequence (string & *key*, void *const *const *data* = *NULL*, int *cursor* = BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

Sequence through a RecordStore, returning the key/data pairs. Sequencing means to start at some point in the store and return the record, then repeatedly calling the sequencor to return the next record. The starting point is typically the the first record, and is set to that when the RecordStore object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

[out] ***key*** The key of the currently sequenced record.

[in] ***data*** Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.

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

Returns

The length of the record currently in sequence.

Exceptions

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::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.15 virtual void BiometricEvaluation::IO::RecordStore::setCursorAtKey (string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[pure virtual]`

Set the sequence cursor to an arbitrary position within the RecordStore, starting at key. Key will be the first record returned from the next call to sequence().

Parameters

`[in]` ***key*** The key of the record which will be returned by the first subsequent call to 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::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.67.3.16 static tr1::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::openRecordStore (const string & *name,* const string & *parentDir,* uint8_t *mode* = ***READWRITE***) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[static]`

Open an existing RecordStore and return a managed pointer to the the object representing that store.

Applications can open existing record stores without the need to know what type of RecordStore it is.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

[in] ***name*** The name of the store to be opened.

[in] ***parentDir*** Where, in the file system, the store is rooted.

[in] ***mode*** The type of access a client of this RecordStore has.

Returns

An object representing the existing store.

Exceptions

Error::ObjectDoesNotExist The RecordStore does not exist.

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

E.67.3.17 static tr1::shared_ptr<RecordStore> BiometricEvaluation::IO::RecordStore::createRecordStore (const string & *name,* const string & *description,* const string & *type,* const string & *destDir*) throw (Error::ObjectExists, Error::StrategyError) [static]

Create a new RecordStore and return a managed pointer to the the object representing that store.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

[in] ***name*** The name of the store to be created.

[in] ***description*** The description of the store to be created.

[in] ***type*** The type of the store to be created.

[in] ***destDir*** Where, in the file system, the store will be created.

Returns

An auto_ptr 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, or the name is malformed.

E.67.3.18 static void BiometricEvaluation::IO::RecordStore::removeRecordStore (const string & *name,* const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]

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

Parameters

[in] ***name*** The name of the existing RecordStore.

[in] ***parentDir*** Where, in the file system, the store is rooted.

Exceptions

Error::ObjectDoesNotExist A record with the given key does not exist.

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

E.67.3.19 static void BiometricEvaluation::IO::RecordStore::mergeRecordStores (const string & *mergedName,* const string & *mergedDescription,* const string & *parentDir,* const string & *type,* RecordStore * *recordStores[ ],* size_t *numRecordStores*) throw (Error::ObjectExists, Error::StrategyError) [static]

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

Parameters

[in] ***mergedName*** The name of the new RecordStore that will be created.

[in] ***mergedDescription*** The text used to describe the RecordStore.

[in] ***parentDir*** Where, in the file system, the new store should be rooted.

[in] ***type*** The type of RecordStore that mergedName should be.

[in] ***recordStores*** An array of RecordStore* that should be merged into mergedName.

[in] ***numRecordStores*** The number of RecordStore* in recordStores.

Exceptions

Error::ObjectExists A RecordStore with mergedNamed in parentDir already exists.

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

E.67.3.20 static void BiometricEvaluation::IO::RecordStore::mergeRecordStores (const string & *mergedName,* const string & *mergedDescription,* const string & *parentDir,* const string & *type,* tr1::shared_ptr< RecordStore > *recordStores[ ],* size_t *numRecordStores*) throw (Error::ObjectExists, Error::StrategyError) [static]

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

Parameters

[in] ***mergedName*** The name of the new RecordStore that will be created.

[in] ***mergedDescription*** The text used to describe the RecordStore.

[in] ***parentDir*** Where, in the file system, the new store should be rooted.

[in] ***type*** The type of RecordStore that mergedName should be.

[in] ***recordStores*** An array of RecordStore shared pointers, such as those returned from IO::Factory, that should be merged into mergedName.

[in] ***numRecordStores*** The number of RecordStore* in recordStores.

Exceptions

Error::ObjectExists A RecordStore with mergedNamed in parentDir already exists.

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

E.67.4 Member Data Documentation

E.67.4.1 const string BiometricEvaluation::IO::RecordStore::CONTROLFILENAME [static]

The name of the control file, a properties list

E.67.4.2 const string BiometricEvaluation::IO::RecordStore::NAMEPROPERTY [static]

Property key for name of the RecordStore

E.67.4.3 const string BiometricEvaluation::IO::RecordStore::DESCRIPTIONPROPERTY [static]

Property key for description of the RecordStore

E.67.4.4 const string BiometricEvaluation::IO::RecordStore::COUNTPROPERTY [static]

Property key for the number of items in the store

E.67.4.5 const string BiometricEvaluation::IO::RecordStore::TYPEPROPERTY [static]

Property key for the type of RecordStore

E.67.4.6 const string BiometricEvaluation::IO::RecordStore::BERKELEYDBTYPE [static]

DBRecordStore type

E.67.4.7 const string BiometricEvaluation::IO::RecordStore::ARCHIVETYPE [static]

ArchiveRecordStore type

E.67.4.8 const string BiometricEvaluation::IO::RecordStore::FILETYPE [static]

FileRecordStore type

E.67.4.9 const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_START = 1 [static]

Tell sequence() to sequence from beginning

E.67.4.10 const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_-SEQ_NEXT = 2 [static]

Tell sequence to sequence from current position

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

- be_io_recordstore.h

E.68 BiometricEvaluation::Image::Resolution Struct Reference

A structure to represent the resolution of an image.

`#include <be_image.h>`

Public Types

- enum Kind { NA = 0, PPI = 1, PPMM = 2, PPCM = 3 }

  Possible representations of the units in a Resolution struct.

Public Member Functions

- Resolution (const double xRes=0.0, const double yRes=0.0, const Kind units=PPI)

  Create a Resolution struct.

Public Attributes

- double xRes
- double yRes
- Kind units

E.68.1 Detailed Description

A structure to represent the resolution of an image.

E.68.2 Member Enumeration Documentation

E.68.2.1 enum BiometricEvaluation::Image::Resolution::Kind

Possible representations of the units in a Resolution struct.

Enumerator:

NA Not-applicable: unknown, or otherwise

PPI Pixels per inch

PPMM Pixels per millimeter

PPCM Pixels per centimeter

E.68.3 Constructor & Destructor Documentation

E.68.3.1 BiometricEvaluation::Image::Resolution::Resolution (const double *xRes* = `0.0`, const double *yRes* = `0.0`, const Kind *units* = `PPI`)

Create a Resolution struct.

Parameters

`[in]` ***xRes*** Resolution along the X-axis

`[in]` ***yRes*** Resolution along the Y-axis

`[in]` ***units*** Units in which xRes and yRes are represented

E.68.4 Member Data Documentation

E.68.4.1 double BiometricEvaluation::Image::Resolution::xRes

Resolution along the X-axis

E.68.4.2 double BiometricEvaluation::Image::Resolution::yRes

Resolution along the Y-axis

E.68.4.3 Kind BiometricEvaluation::Image::Resolution::units

Units in which xRes and yRes are represented

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

- be_image.h

E.69 BiometricEvaluation::Feature::RidgeCountExtractionMethod Class Reference

Enumerate the types of extraction methods for ridge counts.

`#include <be_feature_minutiae.h>`

Public Types

- enum **Kind** { **NonSpecific** = 0, **FourNeighbor** = 1, **EightNeighor** = 2, **Other** = 3 }

E.69.1 Detailed Description

Enumerate the types of extraction methods for ridge counts.

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

- be_feature_minutiae.h

E.70 BiometricEvaluation::Feature::RidgeCountItem Struct Reference

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

`#include <be_feature_minutiae.h>`

Public Member Functions

- RidgeCountItem (RidgeCountExtractionMethod::Kind extraction_method, int index_one, int index_two, int count=0)

  Create a RidgeCountItem struct.

Public Attributes

- RidgeCountExtractionMethod::Kind **extraction_method**
- int **index_one**
- int **index_two**
- int **count**

E.70.1 Detailed Description

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

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

- be_feature_minutiae.h

E.71 BiometricEvaluation::Error::SignalManager Class Reference

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

`#include <be_error_signal_manager.h>`

Public Member Functions

- SignalManager () throw (Error::StrategyError)
- SignalManager (const sigset_t signalSet) throw (Error::ParameterError)
- void setSignalSet (const sigset_t signalSet) throw (Error::ParameterError)
- void clearSignalSet ()
- void setDefaultSignalSet ()
- bool sigHandled ()
- void start () throw (Error::StrategyError)
- void stop () throw (Error::StrategyError)
- void setSigHandled ()
- void clearSigHandled ()

Static Public Attributes

- static bool _canSigJump
- static sigjmp_buf _sigJumpBuf

E.71.1 Detailed Description

A SignalManager object is used to handle signals that come from the operating system. Applications typically do not invoke most methods of a SignalManager, except the setSignalSet(), setDefaultSignalSet(), and sigHandled(). An application wishing to just catch memory errors can simply construct a SignalManager object, and invoke sigHandled() at the end of the signal block to detect whether a signal was handled.

The BEGIN_SIGNAL_BLOCK macro sets up the jump block and tells the SignalManager object to start handling signals. Applications can call either setSignalSet() or setDefaultSignalSet() before invoking these macros to indicate which signals are to be handled.

The END_SIGNAL_BLOCK() macro clears the signal set, so from that point forward application code signals will be handled in the system's default manner until another signal block is created.

A SignalManager is passive (i.e. no signal handlers are installed) until that start() method is called, and becomes passive when stop() is invoked. The signals that are to be handled by the object are 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.

E.71.2 Constructor & Destructor Documentation

E.71.2.1 BiometricEvaluation::Error::SignalManager::SignalManager ( ) throw (Error::StrategyError)

Construct a new SignalManager object with the default signal handling: SIGSEGV and SIGBUS.

Returns

The SignalManager.

Exceptions

Error::StrategyError Could not register the signal handler.

E.71.2.2 BiometricEvaluation::Error::SignalManager::SignalManager (const sigset_t *signalSet*) throw (Error::ParameterError)

Construct a new SignalManager object with the specified signal handling, no defaults.

Parameters

signalSet (in) The signal set; see sigaction(2), sigemptyset(3) and sigaddset(3).

Returns

The SignalManager.

Exceptions

Error::ParameterError One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

E.71.3 Member Function Documentation

E.71.3.1 void BiometricEvaluation::Error::SignalManager::setSignalSet (const sigset_t *signalSet*) throw (Error::ParameterError)

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.).

E.71.3.2 void BiometricEvaluation::Error::SignalManager::clearSignalSet ( )

Clear all signal handling.

E.71.3.3 void BiometricEvaluation::Error::SignalManager::setDefaultSignalSet ( )

Set the default signals this object will manage: SIGSEGV and SIGBUS.

E.71.3.4 bool BiometricEvaluation::Error::SignalManager::sigHandled ( )

Indicate whether a signal was handled.

Returns

true if a signal was handled, false otherwise.

E.71.3.5 void BiometricEvaluation::Error::SignalManager::start ( ) throw (Error::StrategyError)

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.

E.71.3.6 void BiometricEvaluation::Error::SignalManager::stop ( ) throw (Error::StrategyError)

Stop handling signals of the current signal set.

Exceptions

Error::StrategyError Could not register the signal handler.

E.71.3.7 void BiometricEvaluation::Error::SignalManager::setSigHandled ( )

Set a flag to indicate a signal was handled.

E.71.3.8 void BiometricEvaluation::Error::SignalManager::clearSigHandled ( )

Clear the indication that a signal was handled.

E.71.4 Member Data Documentation

E.71.4.1 bool BiometricEvaluation::Error::SignalManager::_canSigJump [static]

Flag indicating can jump after handling a signal.

Note

Should not be directly used by applications.

E.71.4.2 sigjmp_buf BiometricEvaluation::Error::SignalManager::_sigJumpBuf [static]

The jump buffer used by the signal handler.

Note

Should not be directly used by applications.

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

- be_error_signal_manager.h

E.72 BiometricEvaluation::Image::Size Struct Reference

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

`#include <be_image.h>`

Public Member Functions

- Size (const uint32_t xSize=0, const uint32_t ySize=0)

  Create a Size struct.

Public Attributes

- uint32_t xSize
- uint32_t ySize

E.72.1 Detailed Description

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

E.72.2 Constructor & Destructor Documentation

E.72.2.1 BiometricEvaluation::Image::Size::Size (const uint32_t *xSize = 0*, const uint32_t *ySize = 0*)

Create a Size struct.

Parameters

`[in]` ***xSize*** Number of pixels on the X-axis

`[in]` ***ySize*** Number of pixels on the Y-axis

E.72.3 Member Data Documentation

E.72.3.1 uint32_t BiometricEvaluation::Image::Size::xSize

Number of pixels on the X-axis

E.72.3.2 uint32_t BiometricEvaluation::Image::Size::ySize

Number of pixels on the Y-axis

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

- be_image.h

E.73 BiometricEvaluation::Process::Statistics Class Reference

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

`#include <be_process_statistics.h>`

Public Member Functions

- Statistics ()
- Statistics (IO::LogCabinet *const logCabinet) throw (Error::NotImplemented, Error::ObjectExists, Error::StrategyError)
- void getCPUTimes (uint64_t *usertime, uint64_t *systemtime) throw (Error::StrategyError, Error::NotImplemented)
- void getMemorySizes (uint64_t *vmrss, uint64_t *vmsize, uint64_t *vmpeak, uint64_t *vmdata, uint64_t *vmstack) throw (Error::StrategyError, Error::NotImplemented)
- uint32_t getNumThreads () throw (Error::StrategyError, Error::NotImplemented)
- void logStats () throw (Error::ObjectDoesNotExist, Error::StrategyError, Error::NotImplemented)

  Create a snapshot of the current process statistics in the LogSheet created in the LogCabinet.

- void startAutoLogging (uint64_t interval) throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError, Error::NotImplemented)

Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.

- void stopAutoLogging () throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Stop the automatic logging of process statistics.

- void callStatistics_logStats ()

E.73.1 Detailed Description

The Statistics class provides an interface for gathering process statistics, such as memory usage, system time, etc. The information gathered by objects of this class are for the current process, and can optionally be logged to a LogSheet object contained within the provided LogCabinet.

Note

The resolution of a returned value for many methods may not match the resolution allowed by the interface. For example, the operating system my allow for second resolution whereas the interface allows microsecond resolution.

E.73.2 Constructor & Destructor Documentation

E.73.2.1 BiometricEvaluation::Process::Statistics::Statistics ( )

Constructor with no parameters.

E.73.2.2 BiometricEvaluation::Process::Statistics::Statistics (IO::LogCabinet *const *logCabinet*) throw (Error::NotImplemented, Error::ObjectExists, Error::StrategyError)

Construct a Statistics object with the associated LogCabinet.

Parameters

`[in]` ***logCabinet*** The LogCabinet obejct where this object will create a LogSheet to contain the statistic information for the process.

Exceptions

Error::NotImplemented Logging is not supported on this OS. This exception can be thrown when any portion of the statistics gathering cannot be completed.

Error::ObjectExists The LogSheet already exists. This exception should rarely, if ever, occur.

Error::StrategyError Failure to create the LogSheet in the cabinet.

E.73.3 Member Function Documentation

E.73.3.1 void BiometricEvaluation::Process::Statistics::getCPUTimes (uint64_t * *usertime,* uint64_t * *systemtime*) throw (Error::StrategyError, Error::NotImplemented)

Obtain the total user and system times for the process, in microseconds. Any of the out parameters can be NULL, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

`[out]` ***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.

E.73.3.2 void BiometricEvaluation::Process::Statistics::getMemorySizes (uint64_t * *vmrss,* uint64_t * *vmsize,* uint64_t * *vmpeak,* uint64_t * *vmdata,* uint64_t * *vmstack*) throw (Error::StrategyError, Error::NotImplemented)

Obtain the current memory set sizes for the process, in kilobytes. Any of the out parameters can be NULL, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

`[out]` ***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.

E.73.3.3 uint32_t BiometricEvaluation::Process::Statistics::getNumThreads ( ) throw (Error::StrategyError, Error::NotImplemented)

Obtain the number of threads composing this process.

Note

This method may not be implemented in all operating systems.

Exceptions

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.

E.73.3.4 void BiometricEvaluation::Process::Statistics::logStats ( ) throw (Error::ObjectDoesNotExist, Error::StrategyError, Error::NotImplemented)

Create a snapshot of the current process statistics in the LogSheet created in the LogCabinet.

Exceptions

Error::ObjectDoesNotExist The LogSheet does not exist; this object was not created with LogCabinet object.

Error::StrategyError An error occurred when writing to the LogSheet.

Error::NotImplemented The statistics gathering is not implemented for this operating system.

E.73.3.5 void BiometricEvaluation::Process::Statistics::startAutoLogging (uint64_t *interval*) throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError, Error::NotImplemented)

Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.

Note

It is unrealistic to expect that log entries can be made at a rate of one per microsecond.
If stopAutoLogging() is called very soon after the start, a log entry may not be made.

Parameters

`[in]` ***interval*** The gap between logging snapshots, in microseconds.

Exceptions

Error::ObjectDoesNotExist The LogSheet does not exist; this object was not created with LogCabinet object.

Error::ObjectExists Autologging is currently invoked.

Error::StrategyError An error occurred when writing to the LogSheet.

Error::NotImplemented The statistics gathering is not implemented for this operating system.

E.73.3.6 void BiometricEvaluation::Process::Statistics::stopAutoLogging ( ) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Stop the automatic logging of process statistics.

Exceptions

Error::ObjectDoesNotExist Not currently autologging.

Error::StrategyError An error occurred when stopping, most likely because the logging thread died.

E.73.3.7 void BiometricEvaluation::Process::Statistics::callStatistics_logStats ( )

Helper function in C++ space that has access to this object, and is called from C space by the logging thread. Applications should not call this function.

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

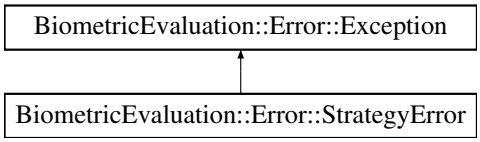
- be_process_statistics.h

E.74 BiometricEvaluation::Error::StrategyError Class Reference

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

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::StrategyError:



Public Member Functions

- StrategyError ()
- StrategyError (string info)

E.74.1 Detailed Description

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

E.74.2 Constructor & Destructor Documentation

E.74.2.1 BiometricEvaluation::Error::StrategyError::StrategyError ( )

Construct a StrategyError object with the default information string.

Returns

The StrategyError object.

E.74.2.2 BiometricEvaluation::Error::StrategyError::StrategyError (string *info*)

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

Returns

The StrategyError object.

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

- be_error_exception.h

E.75 BiometricEvaluation::Time::Timer Class Reference

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

`#include <be_time_timer.h>`

Public Member Functions

- Timer ()
- void start () throw (Error::StrategyError)
- void stop () throw (Error::StrategyError)
- uint64_t elapsed () throw (Error::StrategyError)

E.75.1 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.

E.75.2 Constructor & Destructor Documentation

E.75.2.1 BiometricEvaluation::Time::Timer::Timer ( )

Constructor for the Timer object.

E.75.3 Member Function Documentation

E.75.3.1 void BiometricEvaluation::Time::Timer::start ( ) throw (Error::StrategyError)

Start tracking time.

Exceptions

Error::StrategyError This object is currently timing an operation or an error occurred when obtaining timing information.

E.75.3.2 void BiometricEvaluation::Time::Timer::stop ( ) throw (Error::StrategyError)

Stop tracking time.

Exceptions

Error::StrategyError This object is not currently timing an operation or an error occurred when obtaining timing information.

E.75.3.3 uint64_t BiometricEvaluation::Time::Timer::elapsed ( ) throw (Error::StrategyError)

Get the elapsed time in microseconds between calls to this object's start() and stop() methods.

Returns

The number of microseconds between calls to this object's start() and stop() methods.

Exceptions

Error::StrategyError This object is currently timing an operation or an error occurred when obtaining timing information.

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

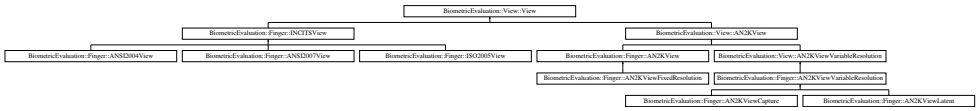
- be_time_timer.h

E.76 BiometricEvaluation::View::View Class Reference

A class to represent single biometric element view.

`#include <be_view_view.h>`

Inheritance diagram for BiometricEvaluation::View::View:



Public Member Functions

- virtual tr1::shared_ptr< Image::Image > getImage () const =0

  Obtain the image used for the finger view.

- virtual Image::Size getImageSize () const =0

  Obtain the image size.

- virtual Image::Resolution getImageResolution () const =0

  Obtain the image resolution.

- virtual uint32_t getImageDepth () const =0

  Obtain the image depth.

- virtual Image::CompressionAlgorithm::Kind getCompressionAlgorithm () const =0

  Obtain the compression algorithm used on the image.

- virtual Image::Resolution getScanResolution () const =0

  Obtain the image scan resolution.

E.76.1 Detailed Description

A class to represent single biometric element view. Included in a view is the biometric image and any derived information, such as minutiae points.

E.76.2 Member Function Documentation

E.76.2.1 virtual tr1::shared_ptr<Image::Image> BiometricEvaluation::View::View::getImage ( ) const [pure virtual]

Obtain the image used for the finger view.

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

Implemented in BiometricEvaluation::Finger::INCITSView, and BiometricEvaluation::View::AN2KView.

E.76.2.2 virtual Image::Size BiometricEvaluation::View::View::getImageSize ( ) const [pure virtual]

Obtain the image size.

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

Implemented in BiometricEvaluation::Finger::INCITSView, and BiometricEvaluation::View::AN2KView.

E.76.2.3 virtual Image::Resolution BiometricEvaluation::View::View::getImageResolution ( ) const [pure virtual]

Obtain the image resolution.

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

Implemented in BiometricEvaluation::Finger::INCITSView, and BiometricEvaluation::View::AN2KView.

E.76.2.4 virtual uint32_t BiometricEvaluation::View::View::getImageDepth ( ) const [pure virtual]

Obtain the image depth.

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

for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

Implemented in BiometricEvaluation::Finger::INCITSView, and BiometricEvaluation::View::AN2KView.

E.76.2.5 virtual Image::CompressionAlgorithm::Kind BiometricEvaluation::View::View::getCompressionAlgorithm ( ) const [pure virtual]

Obtain the compression algorithm used on the image.

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

Implemented in BiometricEvaluation::Finger::INCITSView, and BiometricEvaluation::View::AN2KView.

E.76.2.6 virtual Image::Resolution BiometricEvaluation::View::View::getScanResolution ( ) const [pure virtual]

Obtain the image scan resolution.

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

Implemented in BiometricEvaluation::Finger::INCITSView, and BiometricEvaluation::View::AN2KView.

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

- be_view_view.h

E.77 BiometricEvaluation::Time::Watchdog Class Reference

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

`#include <be_time_watchdog.h>`

Public Member Functions

- Watchdog (const uint8_t type) throw (Error::ParameterError)
- void setInterval (uint64_t interval)
- void start () throw (Error::StrategyError)
- void stop () throw (Error::StrategyError)
- bool expired ()
- void setCanSigJump ()
- void clearCanSigJump ()
- void setExpired ()
- void clearExpired ()

Static Public Attributes

- static const uint8_t PROCESSTIME = 0
- static const uint8_t REALTIME = 1
- static bool **_canSigJump**
- static sigjmp_buf **_sigJumpBuf**

E.77.1 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 there own signal handlers, but use the SignalManager class instead.

The BEGIN_WATCHDOG_BLOCK macro sets up the jump block and tells the Watchdog object to start handling the alarm signal. Applications must call setInterval() before invoking the BEGIN_WATCHDOG_BLOCK() macro.

The END_WATCHDOG_BLOCK() macro disables the watchdog timer, but doesn't affect the current interval value. Applications can set the interval once and use the BEGIN/END block macros repeatedly. Failure to call setInterval() results in an effectively disabled timer, as does setting the interval to 0.

Note

Process virtual timing may not be available on all systems. In those cases, an application compilation error will occur because PROCESSTIME will not be defined.

Attention

On many systems, the sleep(3) call is implemented using alarm signals, the same technique used by the Watchdog class. Therefore, applications should not call sleep(3) inside the Watchdog block; behavior is undefined in that case, but usually results in cancellation of the Watchdog timer.
The setCanSigJump(), clearCanSigJump(), setExpired() and clearExpired() methods are not meant to be used directly by applications, which should use the BEGIN_WATCHDOG_BLOCK()/END_WATCHDOG_BLOCK() macro pair.

See also

Error::SignalManager

E.77.2 Constructor & Destructor Documentation

E.77.2.1 BiometricEvaluation::Time::Watchdog::Watchdog (const uint8_t *type*) throw (Error::ParameterError)

Construct a new Watchdog object.

Parameters

`[in]` ***type*** The type of timer, ProcessTime or RealTime.

Returns

The Watchdog object.

Exceptions

Error::ParameterError The type is invalid.

E.77.3 Member Function Documentation

E.77.3.1 void BiometricEvaluation::Time::Watchdog::setInterval (uint64_t *interval*)

Set the interval for the timer, but don't start the timer. Setting a value of 0 will essentially disable the timer. Timer intervals are in microseconds, however actual intervals are dependent on the resolution of the system clock, and may not be at microsecond resolution.

Parameters

`[in]` ***interval*** The timer interval, in microseconds.

E.77.3.2 void BiometricEvaluation::Time::Watchdog::start ( ) throw (Error::StrategyError)

Start a watchdog timer.

Exceptions

Error::StrategyError Could not register the signal handler, or could not create the timer.

E.77.3.3 void BiometricEvaluation::Time::Watchdog::stop ( ) throw (Error::StrategyError)

Stop a watchdog timer.

Exceptions

Error::StrategyError Could not clear the timer.

E.77.3.4 bool BiometricEvaluation::Time::Watchdog::expired ( )

Indicate whether the watchdog timer expired.

Returns

true if the timer expired, false otherwise.

E.77.3.5 void BiometricEvaluation::Time::Watchdog::setCanSigJump ( )

Indicate that the signal handler can jump into the application code after handling the signal.

E.77.3.6 void BiometricEvaluation::Time::Watchdog::clearCanSigJump ( )

Clears the flag for the Watchdog object to indicate that the signal jump block is no longer valid.

E.77.3.7 void BiometricEvaluation::Time::Watchdog::setExpired ( )

Set a flag to indicate the timer expired.

E.77.3.8 void BiometricEvaluation::Time::Watchdog::clearExpired ( )

Clear the flag indicating the timer expired.

E.77.4 Member Data Documentation

E.77.4.1 const uint8_t BiometricEvaluation::Time::Watchdog::PROCESSTIME = 0 [static]

A Watchdog based on process time.

E.77.4.2 const uint8_t BiometricEvaluation::Time::Watchdog::REALTIME = 1 [static]

A Watchdog based on real (wall clock) time.

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

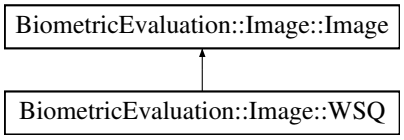
- be_time_watchdog.h

E.78 BiometricEvaluation::Image::WSQ Class Reference

A WSQ-encoded image.

`#include <be_image_wsq.h>`

Inheritance diagram for BiometricEvaluation::Image::WSQ:



Public Member Functions

- **WSQ** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- Memory::AutoArray< uint8_t > getRawData () const throw (Error::DataError)

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

- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)

  Accessor for decompressed data in grayscale.

Static Public Member Functions

- static bool isWSQ (const uint8_t *data)

E.78.1 Detailed Description

A WSQ-encoded image.

E.78.2 Member Function Documentation

E.78.2.1 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::WSQ::getRawData ( ) const throw (Error::DataError) [virtual]

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

Returns

Raw image data.

Exceptions

Error::DataError Error decompressing image data.

Implements BiometricEvaluation::Image::Image.

E.78.2.2 Memory::AutoArray<uint8_t> BiometricEvaluation::Image::WSQ::getRawGrayscaleData (uint8_t *depth* = 8) const throw (Error::DataError, Error::ParameterError) [virtual]

Accessor for decompressed data in grayscale.

Parameters

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

Returns

Raw image buffer.

Exceptions

Error::DataError Error decompressing image data.

Error::ParameterError Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.
This method always returns an image that uses 8 bits to represent a single pixel. depth adjusts the number of pixels used to determine the color of the pixel in the 8 bit container, currently 1 (2 gray levels) or 8 (256 gray levels).

Implements BiometricEvaluation::Image::Image.

E.78.2.3 static bool BiometricEvaluation::Image::WSQ::isWSQ (const uint8_t * *data*) [static]

Whether or not data is a WSQ image.

Parameters

[in] ***data*** The buffer to check.

Returns

true if data appears to be a WSQ image, false otherwise

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

- be_image_wsq.h

Index