BIOMETRIC EVALUATION COMMON FRAMEWORK

PROGRAMMER'S GUIDE
VERSION 0.1

WAYNE SALAMON
GREGORY FIUMARA

IMAGE GROUP
INFORMATION ACCESS DIVISION
INFORMATION TECHNOLOGY LABORATORY

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

JANUARY 21, 2014

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 [13].

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++[15] 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` namespace.

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

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

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

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

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

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

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

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

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

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

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

Chapter 3

Framework

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

Listing 3.1: Using the `Framework` API

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

Chapter 4

Memory

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

4.1 AutoBuffer

The Biometric Evaluation Framework is designed to interoperate with existing C code that has its own memory management techniques, e.g. NIST Biometric Image Software [12]. In these cases, functions exist to allocate and free blocks of memory, and these calls must be made by the applications which use those libraries. To assist BECommon clients that use these existing libraries, the `AutoBuffer` class wraps the C memory management functions, guaranteeing the release of C objects when the AutoBuffer goes out of scope.

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

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

Listing 4.1: Using the `AutoBuffer`

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

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


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

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

4.2 AutoArray

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

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

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

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

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

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

AutoArray is adapted from "c_array " [15, 496].

4.3 IndexedBuffer

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

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

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

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

Listing 4.3: Using the IndexedBuffer

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

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

Chapter 5

Error Handling

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

5.1 Biometric Evaluation Exceptions

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

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

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

5.2 Signal Handling

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

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

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

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

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

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

Listing 5.1: Using the `SignalManger`

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

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

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

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

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

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

Listing 5.2: Specifying Signals to the `SignalManger`

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

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

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

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

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

Chapter 6

Input/Output

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

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

6.1 Utility

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

6.2 Record Management

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

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

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

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

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

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

Listing 6.1: Using a `RecordStore`

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

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

    try {
        uint8_t *theData;

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

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

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

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

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

6.3 Logging

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

The logging classes in the `IO` package provide a straight-forward method for applications to record their progress without the need to manage the low-level 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`

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

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

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

6.5 Compressor

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

Listing 6.4: Using a `Compressor` Object

```
tr1::shared_ptr<IO::Compressor> compressor;
Memory::uint8Array compressedBuffer, largeBuffer = /* ... */;
try {
        compressor = IO::Compressor::createCompressor(Compressor::Kind::GZIP);
        /* Overloaded for all combination of buffer and file */
        compressor->compress("largeInputFile", "compressedOutputFile");
        compressedBuffer = compressor->compress(largeBuffer);
} catch (Error::Exception &e) {
```

```
9         cerr << "Could not compress (" << e.getInfo() << ')' << endl;
10 }
```

Different `Compressor` s may be able to respond to options that tune their operations. These options (and approved values) should be well-documented in the child class, however, a no-argument constructor of a child `Compressor` should automatically set any required options to default values. Setting and retrieving these options is very similar to interacting with a `Properties` object (see Section 6.4 on page 17).

Listing 6.5: Setting `Compressor` Options

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

Chapter 7

Time and Timing

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

7.1 Elapsed Time

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

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

Listing 7.1: Using the `Timer`

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

7.2 Limiting Execution Time

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

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

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

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

Listing 7.2: Using the `Watchdog`

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

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

    Time::Timer timer;

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

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

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

Chapter 8

Process Information

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

8.1 Process Statistics

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

Listing 8.1: Gathering Process Statistics

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

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

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

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

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

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

    // Do some other work
```

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

8.2 Process Management

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

8.2.1 Manager

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

8.2.2 Worker

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

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

Listing 8.3: A Responsible `Worker` Implementation

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

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

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

8.2.3 WorkerController

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

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

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

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

```
1  int
2  main(
3      int argc,
4      char *argv[])
5  {
6          static const uint32_t numEmployees = 3;
7          int status = EXIT_FAILURE;
8
9          tr1::shared_ptr<Process::Manager> shiftLeader(new Process::ForkManager);
10         queue<Customer*> *customers = new queue<Customer*>();
11
12         /* Create Employees (Workers/WorkerControllers) */
13         tr1::shared_ptr<Process::WorkerController> employees[numEmployees];
14         for (uint32_t i = 0; i < numEmployees; i++) {
15                 employees[i] = shiftLeader->addWorker(
16                     tr1::shared_ptr<ResponsibleEmployeeTask>(
17                     new ResponsibleEmployeeTask()));
18
19                 /* Assign employees to each Task */
20                 employees[i]->setParameter("employee",
21                     tr1::shared_ptr<Employee>(new Employee()));
22                 employees[i]->setParameter("customers",
23                     tr1::shared_ptr< queue<Customer*> >(customers);
24         }
25
26         /* Employees start serving customers while shift leader manages */
27         shiftLeader->startWorkers(false);
28
29         /* Customers enter the queue... */
30         queue<Restaurant::AdministrativeTasks> adminTasks;
31         adminTasks.push("Inventory");
32         adminTasks.push("Customer Complaints");
33         adminTasks.push("Clean Dining Room");
34
35         while (shiftLeader->getNumActiveWorkers() != 0) {
36                 shiftLeader->doTask(adminTasks.front());
37                 adminTasks.pop();
38         }
39
40         /* ...end of the day */
41         for (uint32_t i = 0; i < numEmployees; i++)
42                 if (employees[i]->isWorking())
43                         shiftLeader->stopWorker(employees[i]);
44
45         /*
46          * Wait a reasonable amount of time before locking up for the night
47          * (in this case, indefinitely).
48          */
```

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

8.2.4 Communications

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

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

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

Listing 8.5: `Worker` Communication

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

Listing 8.6: `Manager` Communication

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

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

Chapter 9

System

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

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

Listing 9.1: Using the `System` CPU Count Information

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

using namespace BiometricEvaluation;

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

    // perform some application setup ...

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

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

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

Chapter 10

Image

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

10.1 The Image Namespace

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

The same applies to images encapsulated in biometric records. Although some biometic records have fields for image attributes like dimensions and resolution, the corresponding fields of an `Image` class are **not** populated with their contents. The `Image` namespace data types *are* used outside of the namespace, such as in finger views, to retrieve image attributes stored as part of the biometric record. Applications can compare those values against the values within the `Image` object, as in most cases those values are taken directly from the underlying image data. See Chapter 14 on page 45 for more information on image-based biometric records.

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

10.2 The Image Class

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

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

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

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

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

10.3 Raw Image

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

10.4 JPEG

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

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

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

10.5 JPEGL

Similar to `JPEG`, the `JPEGL` class performs `Image` class services for lossless JPEG encoded images. JPEGL decompression is performed by NIST Biometric Image Software 's `libjpegl` [12].

10.6 JPEG2000

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

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

Decompression is provided by the OpenJPEG library (`libopenjpeg`) [11]. `JPEG2000` also provides a static function to test whether or not an image appears to be JPEG 2000-encoded.

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

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

10.7 NetPBM

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

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

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

10.8 PNG

The `PNG` class represents an image encoded according to the PNG image standard [5]. Decompression is provided by `libpng` [14].

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

10.9 WSQ

Images encoded in the WSQ-image standard [16] are represented by the `WSQ` class. The WSQ decompressor found in NIST Biometric Image Software [12], `libwsq`, is used by this class. The class provides a static function to determine whether or not an image appears to be encoded in the WSQ format.

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

Chapter 11

Text

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

Listing 11.1 shows how to use the `split()` function from the `Text` package. `split()` can separate a `string` into tokens delimited by a character, useful for processing comma- or space-separated text files (such files could be produced by a `LogSheet` (Section 6.3 on page 16), for instance). Here, a text file containing metadata for an image is being parsed, perhaps to be passed to the `RawImage` constructor (Section 10.3 on page 34).

Listing 11.1: Tokenizing a `string`

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

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

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

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

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

Listing 11.2: `md5sum` via BECommon

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

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

using namespace std;
using namespace BiometricEvaluation;

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

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

        return (EXIT_SUCCESS);
}
```

Chapter 12

Feature

The `Feature` package contains those items that relate to the representation of biometric features, such as fingerprint minutiae, facial features (eyes, etc.), and related information. Objects of these class types are typically associated with `View` (Chapter 14 on page 45) or `DataInterchange` (Chapter 15 on page 47) objects. For example, a minutiae object is usually obtained from a finger view, which may have been obtained from a data interchange object representing an entire biometric record for an individual.

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

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

12.1 ANSI/NIST Features

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

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

12.2 ISO/INCITS Features

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

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

Chapter 13

Finger

One of the most commonly used biometric source is the fingerprint. Multiple types of information can be derived from a fingerprint, including minutiae and the pattern, such as whorl, etc. The `Finger` package contains the types, classes, and other items that are related to fingers and fingerprints. Objects of the `Finger` classes are typically not used in a stand-alone fashion, but are usually obtained from an object in the `DataInterchage` (Chapter 15 on page 47) package.

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

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

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

13.1 ANSI/NIST Minutiae Data Record

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

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

13.1.1 ANSI/NIST Finger Views

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

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

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

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

Listing 13.1: Using an AN2K Finger View

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

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

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

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

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

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

13.1.2 ISO/INCITS Finger Views

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

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

Listing 13.2: Using an INCITS Finger View

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

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

Chapter 14

View

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

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

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

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

Chapter 15

Data Interchange

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

The design of classes in the DataInterchange package allows applications to create a single object from a biometric record, such as an ANSI/NIST file. After creating this object, the application can retrieve the needed information (such as Finger View s Chapter 13 on page 41) from this object. A typical example would be to retrieve all images from the record and pass them into a function that extracts a biometric template or some other image processing.

15.1 ANSI/NIST Data Records

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

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

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

Listing 15.1: Retrieving ANSI/NIST Finger Captures

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

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

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

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

Listing 15.2: Retrieving ANSI/NIST Latent Records

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

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

15.2 INCITS Data Records

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

15.2.1 Finger Views

Within the BECommon, finger view objects (Section 14) 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 13.2 on page 43 shows an example of accessing the information in an ANSI 378-2004 Finger Minutiae Record by creating an `ANSI2004View` object from the record file.

References

[1] *ANSI INCITS 378-2004: Finger Minutiae Format for Data Interchange*. ANSI/INCITS, 2004. 39, 43, 50

[2] *ISO/IEC 19794-2: Information technology - Biometric data interchange formats - Part 2: Finger minutiae data*. ISO/IEC, first edition, 2005. 39, 50

[3] *American National Standard for Information Systems - Data Format for the Interchange of Fingerprint, Facial, & Other Biometric Information*. ANSI/NIST-ITL, 1-2007 edition, 2007. 4, 39, 41, 47

[4] Mark Adler. zlib, 2012. http://www.zlib.net/. 18

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

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

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

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

[9] International Committee for Information Technology Standards. http://www.incits.org. 43, 50

[10] ISO/IEC Joint Technical Committee 1/SC 37 Biometrics. 43

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

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

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

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

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

[16] 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

API Reference

Appendix B

Namespace Index

B.1 Namespace List

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

Appendix C

Hierarchical Index

C.1 Class Hierarchy

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

Appendix D

Class Index

D.1 Class List

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

Appendix E

Namespace Documentation

E.1 BiometricEvaluation::Error Namespace Reference

Exceptions, and other error handling.

Classes

- class Exception

  The parent class of all BiometricEvaluation exceptions.
- class FileError

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

  An invalid parameter was passed to a constructor or method.
- class ConversionError

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

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

  An error occurred when allocating an object.
- class ObjectExists

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

  The named object does not exist.
- class ObjectIsOpen

  The object is already opened.
- class ObjectIsClosed

  The object is closed.
- class StrategyError

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

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

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

Functions

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

E.1.1 Detailed Description

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

E.1.2 Function Documentation

string BiometricEvaluation::Error::errorStr ( )

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

Returns

The current error message specified by errno.

E.2 BiometricEvaluation::Finger Namespace Reference

Biometric information relating to finger images and derived information.

Classes

- class PatternClassification

  Pattern classification codes.
- class Position

  Finger position codes.
- class Impression

  Finger and palm impression types.
- class FingerImageCode
- class AN2KMinutiaeDataRecord

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

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

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

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

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

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

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

A class to represent single finger view and derived information.

- class ISO2005View

  A class to represent single finger view and derived information.

Typedefs

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

Functions

- std::ostream & operator<< (std::ostream &, const Finger::PatternClassification::Kind &)

  Output stream overload for PatternClassification::Kind.
- std::ostream & **operator**<< (std::ostream &, const Position::Kind &)
- std::ostream & **operator**<< (std::ostream &, const Impression::Kind &)
- std::ostream & **operator**<< (std::ostream &, const FingerImageCode::Kind &)
- std::ostream & operator<< (std::ostream &stream, const AN2KViewCapture::AmputatedBandaged::-Kind &ab)

  Output stream overload for AmputatedBandaged::Kind.
- std::ostream & operator<< (std::ostream &stream, const AN2KViewCapture::FingerSegmentPosition &fsp)

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

  Output stream overload for PrintPositionCoordinate.

E.2.1 Detailed Description

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

E.2.2 Function Documentation

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

Output stream overload for PrintPositionCoordinate.

Parameters

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

Returns

Stream with a ppc textual representation appended.

E.3 BiometricEvaluation::Framework Namespace Reference

Information about the framework.

Functions

- unsigned int getMajorVersion ()

  Framework major version.
- unsigned int getMinorVersion ()

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

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

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

  Time when this framework was compiled.
- std::string getCompilerVersion ()

  Version string of compiler used to compile this framework.

E.3.1 Detailed Description

Information about the framework.

E.3.2 Function Documentation

unsigned int BiometricEvaluation::Framework::getMajorVersion ( )

Framework major version.

Returns

The major version number of the BiometricFramework

unsigned int BiometricEvaluation::Framework::getMinorVersion ( )

Framework minor version.

Returns

The minor version of the BiometricEvaluation framework.

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

Compiler used to compile this framework.

Returns

The name of the compiler used to compile this framework.

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

Date when this framework was compiled.

Returns

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

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

Time when this framework was compiled.

Returns

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

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

Version string of compiler used to compile this framework.

Returns

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

E.4 BiometricEvaluation::Image Namespace Reference

Basic information relating to images.

Classes

- class CompressionAlgorithm

  Image compression algorithms.
- struct Coordinate

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

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

  A structure to represent the resolution of an image.
- class Image

  Represent attributes common to all images.
- class JPEG

  A JPEG-encoded image.
- class JPEG2000

  A JPEG-2000-encoded image.
- class JPEGL

  A Lossless JPEG-encoded image.
- class NetPBM

  A NetPBM-encoded image.
- class PNG

  A PNG-encoded image.

- class Raw

  An image with no encoding or compression.
- class WSQ

  A WSQ-encoded image.

Typedefs

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

Functions

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

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

  Calculate the distance between two points.

E.4.1 Detailed Description

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

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

E.4.2 Function Documentation

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.

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.

E.5 BiometricEvaluation::IO Namespace Reference

Input/Output functionality.

Namespaces

- Utility

Classes

- struct ManifestEntry
- class ArchiveRecordStore

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

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

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

  Compressor for gzip compression from zlib.
- class ListRecordStore

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

  A class to represent a single logging mechanism.
- class Properties

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

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

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

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

Typedefs

- typedef Memory::OrderedMap < string, ManifestEntry > ManifestMap
- typedef map< string, string > PropertiesMap

E.5.1 Detailed Description

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

E.5.2 Typedef Documentation

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

Convenience typedef for storing the manifest

typedef map<string, string> BiometricEvaluation::IO::PropertiesMap

Internal structure used for storing property keys/values

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

Functions

- void removeDirectory (const string &directory, const string &prefix) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Remove a directory using directory name and parent pathname.
- void removeDirectory (const string &pathname) throw (Error::ObjectDoesNotExist, Error::StrategyError)

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

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

  Set aside a file or directory name.
- uint64_t getFileSize (const string &pathname) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- bool fileExists (const string &pathname) throw (Error::StrategyError)
- bool **pathIsDirectory** (const string &pathname) throw (Error::StrategyError)
- bool validateRootName (const string &name)
- bool constructAndCheckPath (const string &name, const string &parentDir, string &fullPath)
- int makePath (const string &path, const mode_t mode)

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

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

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

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

  Determine if a file can be opened with read permission.
- bool isWritable (const string &pathname)

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

- string createTemporaryFile (const string &prefix="", const string &parentDir="/tmp") throw (Error::-FileError, Error::MemoryError)

  Create a temporary file.
- FILE * createTemporaryFile (string &path, const string &prefix="", const string &parentDir="/tmp") throw (Error::FileError, Error::MemoryError)

  Create a temporary file.

E.6.1 Detailed Description

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

E.6.2 Function Documentation

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

Remove a directory using directory name and parent pathname.
Parameters

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

Exceptions

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

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

Remove a directory using a complete pathname.
Parameters

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

Exceptions

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

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

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

Parameters

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

Exceptions

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

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

Set aside a file or directory name.

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

Parameters

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

Exceptions

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

uint64_t BiometricEvaluation::IO::Utility::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.

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.

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.

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.

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.

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

Read the contents of a file into a buffer.
Parameters

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

Returns

Contents of path in a buffer.

Exceptions

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

void BiometricEvaluation::IO::Utility::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.

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.

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

Determine if a file can be opened with read permission.

Parameters

in	*pathname*	Path to the file to check.

Returns

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

Note

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

See Also

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

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

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

Parameters

in	*pathname*	Path to the file to check.

Returns

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

Note

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

See Also

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

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

Create a temporary file.

Parameters

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

Exceptions

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

Returns

Path to temporary file.

Note

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

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

Create a temporary file.

Exclusivity to the file stream is guaranteed.

Parameters

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

Exceptions

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

Returns

Open file stream to path.

Note

Caller must fclose(3) the returned stream.

E.7 BiometricEvaluation::Memory Namespace Reference

Support for memory-related operations.

Classes

- class AutoArray

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

  Manage a memory buffer with an index.

- class OrderedMap
- class OrderedMapIterator
- class OrderedMapConstIterator

Typedefs

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

E.7.1 Detailed Description

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

E.8 BiometricEvaluation::Process Namespace Reference

Process information and controls.

Classes

- class ForkManager

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

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

  An interface for intranode process management classes.
- class POSIXThreadManager

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

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

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

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

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

Typedefs

- typedef map< string,
  tr1::shared_ptr< void > > ParameterList

E.8.1 Detailed Description

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

E.8.2 Typedef Documentation

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

Convenience typedef for parameter lists to child routines

E.9 BiometricEvaluation::System Namespace Reference

Operating system, hardware, etc.

Functions

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

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

  Obtain the amount of real memory in the system.
- double getLoadAverage () throw (Error::NotImplemented)

  Obtain the system load average for the last minute.

E.9.1 Detailed Description

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

E.9.2 Function Documentation

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.

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.

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.

E.10 BiometricEvaluation::Text Namespace Reference

Text processing for string objects.

Functions

- void removeLeadingTrailingWhitespace (string &s)

  Remove lead and trailing white space from a string object.
- string digest (const string &s, const string &digest="md5") throw (Error::MemoryError, Error::NotImplemented, Error::StrategyError)

  Compute the digest of a string.
- string digest (const void *buffer, const size_t buffer_size, const string &digest="md5") throw (Error::MemoryError, Error::NotImplemented, Error::StrategyError)

  Compute the digest of a memory buffer.
- vector< string > split (const string &str, const char delimiter, bool escape=true) throw (Error::ParameterError)

  Return tokens bound by delimiters and the beginning and end of a string.
- string filename (const string &path)

  Extract the filename portion of a pathname.
- string dirname (const string &path)

  Extract the directory part of a pathname.

E.10.1 Detailed Description

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

E.10.2 Function Documentation

string BiometricEvaluation::Text::digest (const string & *s*, const string & *digest* = `"md5"`) throw Error::MemoryError, Error::NotImplemented, 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.

Exceptions

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

Returns

An ASCII representation of the hex digits composing the digest.

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

Exceptions

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

Returns

An ASCII representation of the hex digits composing the digest.

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

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

Parameters

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

Returns

Vector of string tokens, in order of appearance.

Note

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

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.

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.

E.11 BiometricEvaluation::Time Namespace Reference

Support for time and timers.

Classes

- class Timer

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

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

Functions

- string getCurrentTime ()

  Return the current time as a string.
- void **WatchdogSignalHandler** (int signo, siginfo_t *info, void *uap)

Variables

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

E.11.1 Detailed Description

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

E.12 BiometricEvaluation::View Namespace Reference

View information.

Classes

- class AN2KView

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

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

  A class to represent single biometric element view.

Functions

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

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

  Output stream overload for AN2KQualityMetric.

E.12.1 Detailed Description

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

E.12.2 Function Documentation

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

Output stream overload for AN2KQualityMetric.
Parameters

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

Returns

stream with a qm textual representation appended.

Appendix F

Class Documentation

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

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

`#include <be_finger_an2kview_capture.h>`

Public Types

- enum Kind { Amputated, Bandaged, NA }

F.1.1 Detailed Description

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

F.1.2 Member Enumeration Documentation

enum BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged::Kind

Enumerator

Amputated Amputation

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

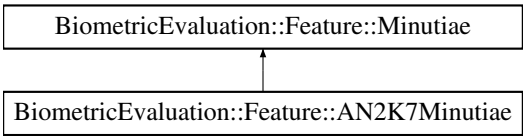
NA Optional field – not specified

F.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 (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 Finger::PatternClassification::Kind convertPatternClassification (const char *fpc) throw (Error::DataError)

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

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

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

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

  Obtain a Coordinate given an AN2K entry.

F.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.

F.2.2 Constructor & Destructor Documentation

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.

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

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

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

Parameters

in	*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.

F.2.3 Member Function Documentation

static Finger::PatternClassification::Kind BiometricEvaluation::Feature::AN2K7-Minutiae::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.

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

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

in	*entry*	A standard pattern classification entry

Exceptions

Error::DataError	Non-standard pattern classification entry.

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.

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.

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.

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	*calculate-Distance*	Whether or not to calculate the [xy]Distance portion of the Coordinate.

Returns

Image::Coordinate representation of str.

Exceptions

Error::DataError	Invalid format of str.

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

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

`#include <be_finger_an2kminutiae_data_record.h>`

Public Member Functions

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

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

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

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

- 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.

Public Attributes

- tr1::shared_ptr
  < Feature::AN2K7Minutiae > const

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

- Impression::Kind const

  Return impression type field from Type-9 Record.

F.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).

F.3.2 Constructor & Destructor Documentation

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.

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

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

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

Parameters

in	*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.

F.3.3 Member Function Documentation

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.

F.3.4 Member Data Documentation

tr1::shared_ptr<Feature::AN2K7Minutiae> BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::const

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

Returns

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

Impression::Kind BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::const

Return impression type field from Type-9 Record.

Returns

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

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

A structure to represent an AN2K quality metric.

`#include <be_view_an2kview_varres.h>`

Public Attributes

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

F.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.

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

A class to represent an entire ANSI/NIST record.

`#include <be_data_interchange_an2k.h>`

Classes

- struct CharacterSet
- struct DomainName

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

Public Types

- typedef struct DomainName DomainName
- typedef struct CharacterSet CharacterSet

Public Member Functions

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

  Constructor taking an AN2K record from a file.
- AN2KRecord (Memory::uint8Array &buf) throw (Error::DataError)

  Constructor taking an AN2K record from a buffer.
- string getVersionNumber () const
- string getDate () const
- string getDestinationAgency () const
- string getOriginatingAgency () const
- string getTransactionControlNumber () const
- string getNativeScanningResolution () const
- string getNominalTransmittingResolution () const
- uint32_t getFingerLatentCount () const

  Obtain the count of latent (Type-13) finger views.
- std::vector < Finger::AN2KViewLatent > getFingerLatents () const

  Obtain all latent (Type-13) finger views.
- uint32_t getFingerCaptureCount () const

  Obtain the count of capture (Type-14) finger views.
- std::vector < Finger::AN2KViewCapture > getFingerCaptures () const

  Obtain all capture (Type-14) finger views.

Static Public Member Functions

- static set< int > recordLocations (Memory::uint8Array &buf, const View::AN2KView::RecordType::-Kind recordType) throw (Error::DataError)

  Find the position within a buffer of all Records of a particular type.
- static set< int > recordLocations (const ANSI_NIST *an2k, const View::AN2KView::RecordType::Kind recordType)

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

Public Attributes

- std::vector < Finger::AN2KMinutiaeDataRecord > const

  Obtain all minutiae (Type-9) data.
- uint8_t const

  Obtain the urgency with which a response is required.
- DomainName const

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

  Obain the date and time of encoding in terms of GMT units.
- std::vector< CharacterSet > const

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

F.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.

F.5.2 Member Typedef Documentation

typedef struct DomainName BiometricEvaluation::DataInterchange::AN2KRecord::DomainName

Convenience typedef for struct DomainName

typedef struct CharacterSet BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet

Convenience typedef for struct CharacterSet

F.5.3 Constructor & Destructor Documentation

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.

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

Constructor taking an AN2K record from a buffer.

Parameters

in	*buf*	The memory buffer containing the complete ANSI/NIST record.

Exceptions

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

F.5.4 Member Function Documentation

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

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

Parameters

in	*buf*	AN2K Buffer to search.
in	*recordType*	The ID of the Record to search for.

Returns

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

Exceptions

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

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

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

Parameters

in	*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.

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

Returns

The record version field in the Type-1 record.

string BiometricEvaluation::DataInterchange::AN2KRecord::getDate ( ) const

Returns

The date field in the Type-1 record.

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

Returns

The destination agency ID.

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

Returns

The originating agency ID.

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

Returns

The transcantion control number.

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

Returns

The native scanning resolution.

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

Returns

The nominal transmitting resolution.

uint32_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerLatentCount ( ) const

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

Returns

The number of latents in the AN2K record.

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

Obtain all latent (Type-13) finger views.
The returned vector will be empty when no latent views are present in the AN2KRecord.

Returns

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

uint32_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerCaptureCount ( ) const

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

Returns

The number of captures in the AN2K record.

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

Obtain all capture (Type-14) finger views.
The returned vector will be empty when no capture views are present in the AN2KRecord.

Returns

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

F.5.5 Member Data Documentation

std::vector<Finger::AN2KMinutiaeDataRecord> BiometricEvaluation::DataInterchange::AN2K-Record::const

Obtain all minutiae (Type-9) data.

Returns

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

uint8_t BiometricEvaluation::DataInterchange::AN2KRecord::const

Obtain the urgency with which a response is required.

Returns

Priority (1:High - 9:Low)

DomainName BiometricEvaluation::DataInterchange::AN2KRecord::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).

struct tm BiometricEvaluation::DataInterchange::AN2KRecord::const

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

Returns

struct tm encoding of the GMT field.

std::vector<CharacterSet> BiometricEvaluation::DataInterchange::AN2KRecord::const

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

Returns

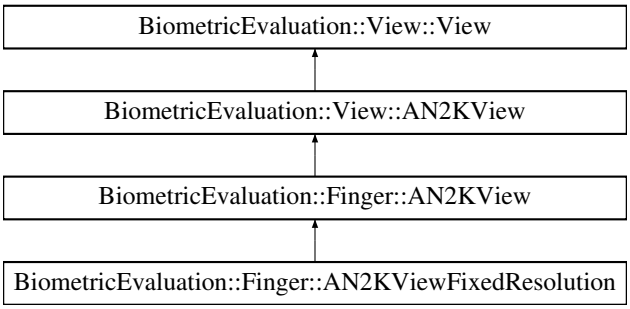
Vector of CharacterSet structs representing other character sets that may appear in the transaction.

F.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 > 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 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 throw (Error::DataError)

  Convert an impression code from a string.
- static Finger::FingerImageCode::Kind convertFingerImageCode (const char *str) throw (Error::DataError)

  Convert an finger image code from a string.

Protected Member Functions

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

  Construct an AN2K finger view from a file.
- AN2KView (Memory::uint8Array &buf, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

  Construct an AN2K finger view from a buffer.
- void addMinutiaeDataRecord (Finger::AN2KMinutiaeDataRecord &mdr)

  Add a minutiae data record to the AN2KMinutiaeDataRecord set.
- void setPositions (Finger::PositionSet &ps)

  Add a position set to the collection of position sets.
- void setImpressionType (Finger::Impression::Kind &imp)

  Mutator for the impression type.

Additional Inherited Members

F.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.

F.6.2 Constructor & Destructor Documentation

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

Construct an AN2K finger view from a file.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.
Error::FileError	An error occurred when reading the file.

BiometricEvaluation::Finger::AN2KView::AN2KView (Memory::uint8Array & *buf,* const RecordType::Kind *typeID,* const uint32_t *recordNumber*) throw Error::ParameterError, Error::DataError) `[protected]`

Construct an AN2K finger view from a buffer.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.

F.6.3 Member Function Documentation

static Finger::Position::Kind BiometricEvaluation::Finger::AN2KView::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.

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.

static Finger::FingerImageCode::Kind BiometricEvaluation::Finger::AN2K-View::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.

vector<AN2KMinutiaeDataRecord> const BiometricEvaluation::Finger::AN2KView::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.

Returns

The vector of minutiae data records.

Finger::PositionSet BiometricEvaluation::Finger::AN2KView::getPositions ( ) const

Obtain the set of finger positions.

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

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

Obtain the finger impression code.

Returns

The finger impression code.

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

Add a minutiae data record to the AN2KMinutiaeDataRecord set.

Parameters

in	*mdr*	The minutiae data record to be added.

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

Add a position set to the collection of position sets.

Parameters

in	*ps*	The position set to be added.

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

Mutator for the impression type.

Parameters

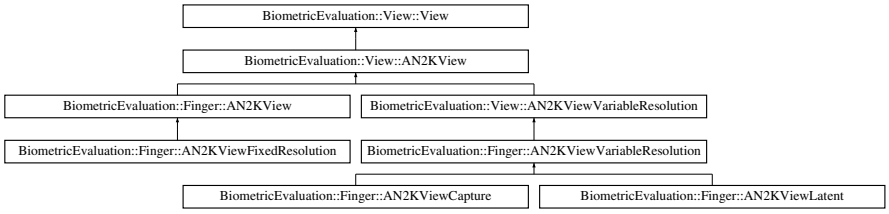
in	*imp*	The impression type for this finger view.

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

A class to represent single biometric view and derived information.

`#include <be_view_an2kview.h>`

Inheritance diagram for BiometricEvaluation::View::AN2KView:



Classes

- class DeviceMonitoringMode

  The level of human monitoring for the image capture device.
- class RecordType

  The type of AN2K record.

Public Member Functions

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

  Construct an AN2K view from a file.
- AN2KView (Memory::uint8Array &buf, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

  Construct an AN2K view from a buffer.
- tr1::shared_ptr< Image::Image > getImage () const

  Obtain the image used for the finger view.
- Image::Size getImageSize () const

  Obtain the image size.
- Image::Resolution getImageResolution () const

  Obtain the image resolution.
- uint32_t getImageDepth () const

  Obtain the image depth.
- Image::CompressionAlgorithm::Kind getCompressionAlgorithm () const

  Obtain the compression algorithm used on the image.
- Image::Resolution getScanResolution () const

  Obtain the image scan resolution.
- vector
  < Finger::AN2KMinutiaeDataRecord >
  const throw (Error::DataError)

  Obtain the set of minutiae records.
- RecordType::Kind getRecordType () const

  Obtain the ANSI-NIST record type.

Static Public Member Functions

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

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

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

Static Public Attributes

- static const double MinimumScanResolutionPPMM

  Constants to define the minimum resolution used for fingerprint images in an AN2k record.
- static const double **HalfMinimumScanResolutionPPMM**
- static const int FixedResolutionBitDepth = 8

  The defined bit-depth for fixed-resolution images.

Protected Member Functions

- 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.

Protected Attributes

- Memory::AutoBuffer< ANSI_NIST > const

  Obtain the complete ANSI/NIST record set.
- RECORD * const

  Obtain a pointer to the single ANSI/NIST record.

F.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.

F.7.2 Constructor & Destructor Documentation

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

Construct an AN2K view from a file.

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

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

Construct an AN2K view from a buffer.

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

F.7.3 Member Function Documentation

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.

static Image::CompressionAlgorithm::Kind BiometricEvaluation::View::AN2KView::convertCompressionAlgorithm (const uint16_t *recordType*, const unsigned char * *an2kValue*) throw Error::ParameterError, Error::DataError) [static]

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

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

Returns

The compression algorithm.

Exceptions

Error::DataError	Invalid compression algorithm for record type.
Error::ParameterError	Invalid record type.

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.

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.

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.

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.

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.

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.

vector<Finger::AN2KMinutiaeDataRecord> const BiometricEvaluation::View::AN2KView::throw (Error::DataError)

Obtain the set of minutiae records.

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

Returns

A vector of minutiae data records.

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

Obtain the ANSI-NIST record type.

Returns

The type of record used to construct this object.

F.7.4 Member Data Documentation

RECORD* BiometricEvaluation::View::AN2KView::const [protected]

Obtain a pointer to the single ANSI/NIST record.

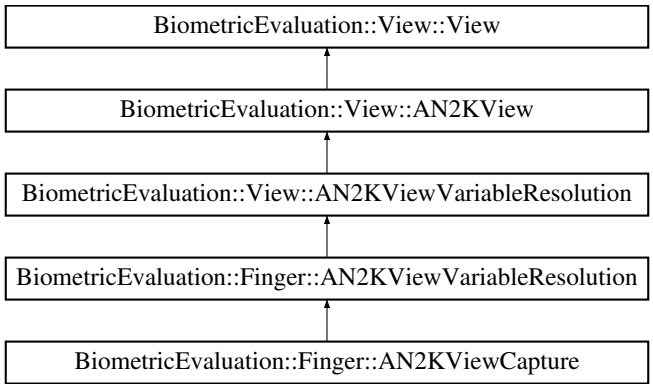
Child classes use this method to obtain a pointer to the specific ANSI/NIST record that was searched for by this class object.

F.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

  Enumeration of the finger amuptated or bandaged code, a reason that a capture could not be made.
- struct FingerSegmentPosition

  Locations of an individual finger segment in a slap.

Public Types

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

Public Member Functions

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

  Construct an AN2K finger view from a file.
- AN2KViewCapture (Memory::uint8Array &buf, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

  Construct an AN2K finger view using from a memory buffer.
- QualityMetricSet extractNISTQuality (const FIELD *field) throw (Error::DataError)

  Extract the NQM information from an AN2K FIELD.

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.

Public Attributes

- PositionDescriptors const

  Return search position descriptors.

- QualityMetricSet const

  Obtain the NIST quality metric for all segmented finger images.

- AmputatedBandaged::Kind const
- FingerSegmentPositionSet const

Additional Inherited Members

F.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).

F.8.2 Constructor & Destructor Documentation

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.

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

Construct an AN2K finger view using from a memory buffer.

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

F.8.3 Member Function Documentation

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.

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.

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.

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.

F.8.4 Member Data Documentation

QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::const

Return search position descriptors.
Obtain metrics for fingerprint image quality score data for the image stored in this record.
Obtain the segmentation quality metric for all segmented finger images.

Returns

QualityMetricSet containing the segmentation quality metric for all segmented finger images.
Optional set of polygonal finger segment positions for all finger segments.
Fingerprint quality metrics

QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::const

Obtain the NIST quality metric for all segmented finger images.
Obtain metrics for fingerprint image quality score data for the image stored in this record.
Obtain the segmentation 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.
```

Returns

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

AmputatedBandaged::Kind BiometricEvaluation::Finger::AN2KViewCapture::const

Returns

Optional amputated or bandaged code.

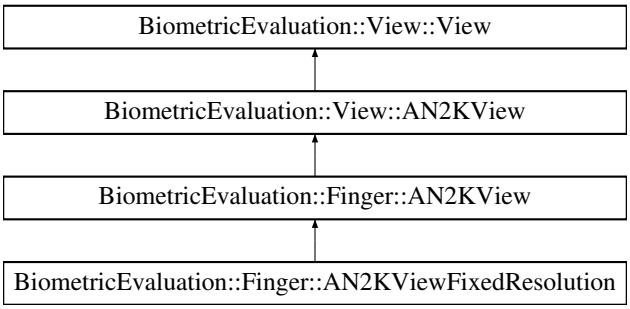
FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::const

Returns

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

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

A class to represent single finger view and derived information.
`#include <be_finger_an2kview_fixedres.h>`
Inheritance diagram for BiometricEvaluation::Finger::AN2KViewFixedResolution:



Public Member Functions

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

  Construct an AN2K finger view from a file.
- AN2KViewFixedResolution (Memory::uint8Array &buf, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

  Construct an AN2K finger view from a buffer.

Additional Inherited Members

F.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.

F.9.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution (const std::string *filename,* const RecordType::Kind *typeID,* const uint32_t *recordNumber*) throw Error::ParameterError, Error::DataError, Error::FileError)

Construct an AN2K finger view from a file.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.
Error::FileError	An error occurred when reading the file.

BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution (Memory::uint8Array & *buf*, const RecordType::Kind *typeID*, const uint32_t *recordNumber*) throw Error::ParameterError, Error::DataError)

Construct an AN2K finger view from a buffer.

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

Parameters

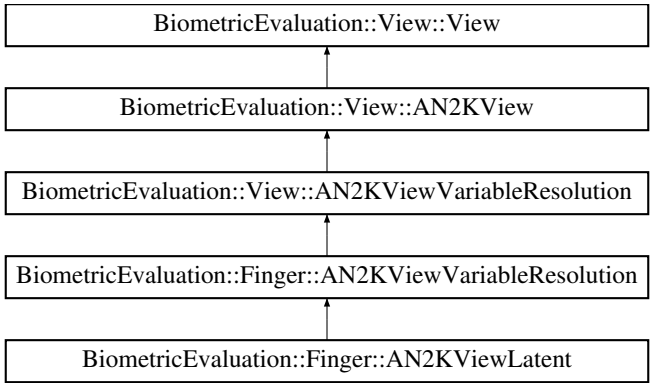
in	*buf*	The buffer containing the AN2K record.
in	*typeID*	The type of AN2K finger view: Type-3/Type-4/etc.
in	*recordNumber*	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.

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

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewLatent:



Public Member Functions

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

  Construct an AN2K finger view from a file.

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

  Construct an AN2K finger view using from a memory buffer.

Public Attributes

- QualityMetricSet const

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

- PositionDescriptors const

  Return search position descriptors.

Additional Inherited Members

F.10.1 Constructor & Destructor Documentation

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.

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

Construct an AN2K finger view using from a memory buffer.

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

F.10.2 Member Data Documentation

QualityMetricSet BiometricEvaluation::Finger::AN2KViewLatent::const

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

Returns

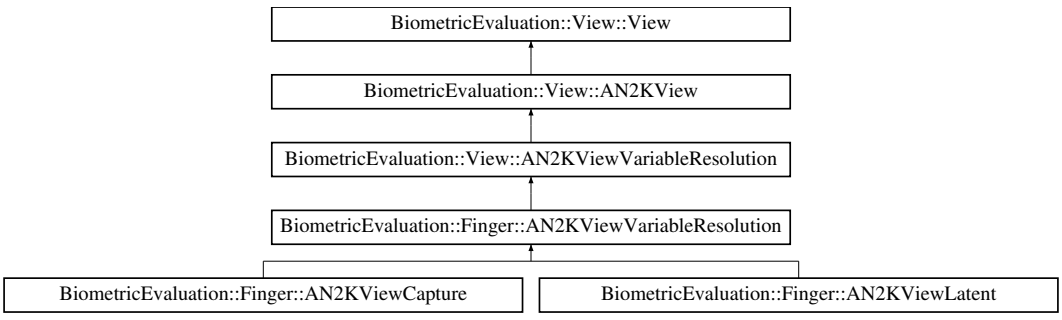
Latent quality metrics

F.11 BiometricEvaluation::Finger::AN2KViewVariableResolution Class Reference

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

`#include <be_finger_an2kview_varres.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewVariableResolution:



Classes

- struct PrintPositionCoordinate

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

Public Types

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

Public Member Functions

- Finger::PositionSet getPositions () const

  Obtain the set of finger positions.
- Finger::Impression::Kind getImpressionType () const

Public Attributes

- PrintPositionCoordinateSet const

  Obtain print position coordinates.

Protected Member Functions

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

  Construct an AN2K finger view from a file.
- AN2KViewVariableResolution (Memory::uint8Array &buf, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

  Construct an AN2K finger view from a buffer.

Static Protected Member Functions

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

  Convert a print position coordinate AN2K subfield to a PrintPositionCoordinate object.
- static PositionDescriptors parsePositionDescriptors (const RecordType::Kind typeID, const RECORD *record) throw (Error::DataError)

  Parse position descriptors from a record.

Protected Attributes

- PositionDescriptors const

Additional Inherited Members

F.11.1 Detailed Description

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

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

F.11.2 Constructor & Destructor Documentation

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

Construct an AN2K finger view from a file.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.
Error::FileError	An error occurred when reading the file.

BiometricEvaluation::Finger::AN2KViewVariableResolution::AN2KViewVariableResolution (Memory::uint8Array & *buf,* const RecordType::Kind *typeID,* const uint32_t *recordNumber*) throw Error::ParameterError, Error::DataError) [protected]

Construct an AN2K finger view from a buffer.

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

Parameters

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

Exceptions

Error::ParameterError	An invalid parameter was passed in.
Error::DataError	An error occurred when parsing the AN2K record.

F.11.3 Member Function Documentation

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.

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

Returns

The finger impression code.

static PrintPositionCoordinate BiometricEvaluation::Finger::AN2KViewVariableResolution-::convertPrintPositionCoordinate (SUBFIELD * *subfield*) throw Error::DataError) [static], [protected]

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

Parameters

in	*subfield*	A print position coordinate AN2K subfield

Returns

Object representation of field.

Exceptions

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

static PositionDescriptors BiometricEvaluation::Finger::AN2KViewVariableResolution::parsePosition-Descriptors (const RecordType::Kind *typeID,* const RECORD * *record*) throw Error::DataError) [static], [protected]

Parse position descriptors from a record.

Parameters

in	*typeID*	The logical record type.
in	*record*	The opened AN2K record.

Returns

Mapping of finger position codes to finger image code.

F.11.4 Member Data Documentation

PrintPositionCoordinateSet BiometricEvaluation::Finger::AN2KViewVariableResolution::const

Obtain print position coordinates.

Returns

Set of all PrintPositionCoordinates

PositionDescriptors BiometricEvaluation::Finger::AN2KViewVariableResolution::const [protected]

Returns

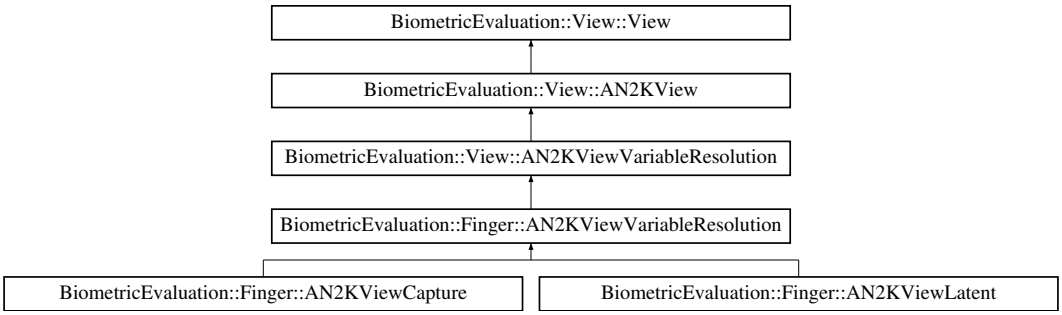
The set of position descriptors.

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

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

`#include <be_view_an2kview_varres.h>`

Inheritance diagram for BiometricEvaluation::View::AN2KViewVariableResolution:



Classes

- struct AN2KQualityMetric

  A structure to represent an AN2K quality metric.

Public Types

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

Public Member Functions

- string getSourceAgency () const
- string getCaptureDate () const
- 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 throw (Error::DataError)

  Read a Quality Metric Set from a variable resolution AN2K record.
- static Memory::uint8Array parseUserDefinedField (const RECORD *const record, int fieldID) throw (Error::ParameterError)

  Read raw bytes from a user-defined AN2K field.

Protected Member Functions

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

  Construct an AN2K finger view from a file.
- AN2KViewVariableResolution (Memory::uint8Array &buf, const RecordType::Kind typeID, const uint32_t recordNumber) throw (Error::ParameterError, Error::DataError)

  Construct an AN2K finger view using from a memory buffer.

Protected Attributes

- QualityMetricSet const

  Obtain quality metrics for associated image record.

Additional Inherited Members

F.12.1 Detailed Description

A class to represent single view based on an ANSI/NIST record.
The view represents a variable resolution (Type-13/14/15) AN2K record.

F.12.2 Constructor & Destructor Documentation

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

Construct an AN2K finger view from a file.
The file must contain the entire AN2K record, not just the finger image and/or minutiae records.

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

Construct an AN2K finger view using from a memory buffer.
The buffer must contain the entire AN2K record, not just the finger image and/or minutiae records.

F.12.3 Member Function Documentation

static QualityMetricSet BiometricEvaluation::View::AN2KViewVariableResolution::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.

Exceptions

Error::DataError	The data contains an invalid value.

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

Returns

The source agency.

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

Returns

The capture date.

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.

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.

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

Read raw bytes from a user-defined AN2K field.

Parameters

in	*record*	Pointer to a RECORD containing the user-defined field.

in	*fieldID*	The user-defined field number.

Returns

Raw bytes from field.

Exceptions

Error::ParameterError	Invalid value for fieldID.

F.12.4 Member Data Documentation

QualityMetricSet BiometricEvaluation::View::AN2KViewVariableResolution::const [protected]

Obtain quality metrics for associated image record.

Returns

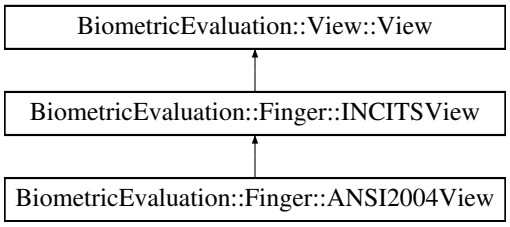
Quality metrics

F.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 (Memory::uint8Array &fmrBuffer, 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.

Additional Inherited Members

F.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.

F.13.2 Constructor & Destructor Documentation

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.

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

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

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

Parameters

in	*fmrBuffer*	The buffer containing the complete finger minutiae record.

in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

F.13.3 Member Function Documentation

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.

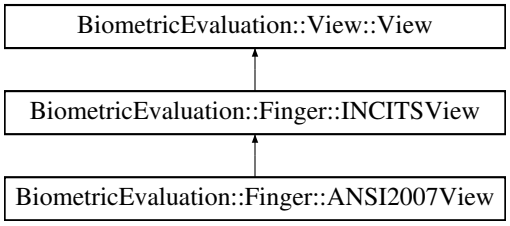
Implements BiometricEvaluation::Finger::INCITSView.

F.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 (Memory::uint8Array &fmrBuffer, 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)
- void **readFVMR** (Memory::IndexedBuffer &buf) throw (Error::DataError)
- virtual void readCoreDeltaData (Memory::IndexedBuffer &buf, uint32_t dataLength, Feature::CorePoint-Set &cores, Feature::DeltaPointSet &deltas) throw (Error::DataError)

  Read the core points data.

Additional Inherited Members

F.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.

F.14.2 Constructor & Destructor Documentation

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.

Exceptions

Error::DataError	Invalid record format.

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

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

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

Parameters

in	*fmrBuffer*	The buffer containing the complete finger minutiae record.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.

F.14.3 Member Function Documentation

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.

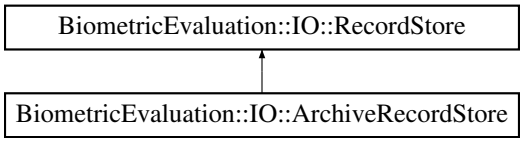
Implements BiometricEvaluation::Finger::INCITSView.

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

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

`#include <be_io_archiverecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ArchiveRecordStore:



Public Member Functions

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

  Obtain real storage utilization.
- void sync () const throw (Error::StrategyError)

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

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- bool needsVacuum ()
- string getArchiveName () const
- string getManifestName () const

Static Public Member Functions

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

Static Public Attributes

- static const long OFFSET_RECORD_REMOVED = -1

Additional Inherited Members

F.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.

F.15.2 Constructor & Destructor Documentation

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.

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.

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

Destructor.

F.15.3 Member Function Documentation

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

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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 inserted.
in	*data*	The data for the record.
in	*size*	The size, in bytes, of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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.

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

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

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

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

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ArchiveRecordStore::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.

uint64_t BiometricEvaluation::IO::ArchiveRecordStore::length (const string & *key*) const 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.

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

Commit the record's data to storage.

Parameters

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

Exceptions

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

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

Implements BiometricEvaluation::IO::RecordStore.

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

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

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

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ArchiveRecordStore::setCursorAtKey (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.

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.

bool BiometricEvaluation::IO::ArchiveRecordStore::needsVacuum ( )

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

Returns

true if vacuum() would be beneficial false otherwise

static bool BiometricEvaluation::IO::ArchiveRecordStore::needsVacuum (const 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

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.

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

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

Returns

Path to archive file.

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.

F.15.4 Member Data Documentation

const long BiometricEvaluation::IO::ArchiveRecordStore::OFFSET_RECORD_REMOVED = -1 [static]

Offset placeholder indicating a removed record

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

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

`#include <be_memory_autoarray.h>`

Public Types

- typedef T value_type
- typedef size_t size_type
- typedef T * iterator
- typedef const T * const_iterator
- typedef T & reference
- typedef const T & const_reference

Public Member Functions

- operator T * ()

  Convert AutoArray to T array.
- operator const T * () const

  Convert AutoArray to const T array.
- reference operator[ ] (ptrdiff_t index)

Subscripting operator overload with unchecked access.

- const_reference operator[ ] (ptrdiff_t index) const

  Const subscripting operator overload with unchecked access.
- reference at (ptrdiff_t index) throw (out_of_range)

  Subscript into the AutoArray with checked access.
- const_reference at (ptrdiff_t index) const throw (out_of_range)

  Subscript into the AutoArray with checked access.
- iterator begin ()

  Obtain an iterator to the beginning of the AutoArray.
- iterator end ()

  Obtain an iterator to the end of the AutoArray.
- void resize (size_type new_size, bool free=false) throw (Error::MemoryError)

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

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

  Deep-copy the contents of a buffer into this AutoArray.
- void swap (AutoArray &first, AutoArray &second)

  Swap two AutoArrays.
- void swap (AutoArray &other)

  Swap this AutoArray with other.
- AutoArray (size_type size=0) throw (Error::MemoryError)

  Construct an AutoArray.
- AutoArray (const AutoArray &copy) throw (Error::MemoryError)

  Construct an AutoArray.
- AutoArray & operator= (AutoArray other) throw (Error::MemoryError)

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

Public Attributes

- const_iterator const

  Obtain an iterator to the beginning of the AutoArray.
- size_type const

  Obtain the number of accessible elements.

F.16.1 Detailed Description

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

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

F.16.2 Member Typedef Documentation

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

Type of element

template<class T> typedef size_t BiometricEvaluation::Memory::AutoArray< T >::size_type

Type of subscripts, counts, etc.

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

Iterator of element

template<class T> typedef const T* BiometricEvaluation::Memory::AutoArray< T >::const_iterator

Const iterator of element

template<class T> typedef T& BiometricEvaluation::Memory::AutoArray< T >::reference

Reference to element

template<class T> typedef const T& BiometricEvaluation::Memory::AutoArray< T >::const_reference

Const reference element

F.16.3 Constructor & Destructor Documentation

template<class T > BiometricEvaluation::Memory::AutoArray< T >::AutoArray (size_type *size = 0*) throw Error::MemoryError)

Construct an AutoArray.

Parameters

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

Exceptions

Error::MemoryError	Could not allocate new memory.

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

Construct an AutoArray.

Parameters

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

Exceptions

Error::MemoryError	Could not allocate new memory.

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

Destructor

F.16.4 Member Function Documentation

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

Convert AutoArray to T array.

Returns

Pointer to the beginning of the underlying array storage.

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

Convert AutoArray to const T array.

Returns

Const pointer to the beginning of the underlying array storage.

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

Subscripting operator overload with unchecked access.
Parameters

in	*index*	Subscript into underlying storage.

Returns

Reference to the element at the specified index.

template<class T > BiometricEvaluation::Memory::AutoArray< T >::const_reference BiometricEvaluation::Memory::AutoArray< T >::operator[ ] (ptrdiff_t *index*) const

Const subscripting operator overload with unchecked access.
Parameters

in	*index*	Subscript into underlying storage.

Returns

Const reference to the element at the specified index.

template<class T > BiometricEvaluation::Memory::AutoArray< T >::reference BiometricEvaluation::Memory::AutoArray< T >::at (ptrdiff_t *index*) throw out_of_range)

Subscript into the AutoArray with checked access.
Parameters

in	*index*	Subscript into underlying storage.

Returns

Reference to the element at the specified index.

Exceptions

out_of_range	Specified index is outside the bounds of this AutoArray.

template<class T > BiometricEvaluation::Memory::AutoArray< T >::const_reference BiometricEvaluation::Memory::AutoArray< T >::at (ptrdiff_t *index*) const throw out_of_range)

Subscript into the AutoArray with checked access.

Parameters

index	Subscript into underlying storage.

Returns

Const reference to the element at the specified index.

Exceptions

out_of_range	Specified index is outside the bounds of this AutoArray.

template<class T > BiometricEvaluation::Memory::AutoArray< T >::const_iterator BiometricEvaluation::Memory::AutoArray< T >::begin ( )

Obtain an iterator to the beginning of the AutoArray.

Returns

Iterator positioned at the first element of the AutoArray.

template<class T > BiometricEvaluation::Memory::AutoArray< T >::const_iterator BiometricEvaluation::Memory::AutoArray< T >::end ( )

Obtain an iterator to the end of the AutoArray.

Returns

Iterator positioned at the one-past-last element of the AutoArray.

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

Change the number of accessible elements.

Parameters

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

Exceptions

Error::MemoryError	Problem allocating memory.

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

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

Parameters

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

Warning

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

template<class T > void BiometricEvaluation::Memory::AutoArray< T >::copy (const_iterator *buffer*, size_type *size*)

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

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

Warning

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

template<class T > void BiometricEvaluation::Memory::AutoArray< T >::swap (AutoArray< T > & *first*, AutoArray< T > & *second*)

Swap two AutoArrays.
Parameters

in/out]	first AutoArray that will become second.
in/out]	second AutoArray that will become first.

template<class T > void BiometricEvaluation::Memory::AutoArray< T >::swap (AutoArray< T > & *other*)

Swap this AutoArray with other.
Parameters

in/out]	other AutoArray that will become this.

Note

Mainly for use when called from std::swap total template specialization.

template<class T > BiometricEvaluation::Memory::AutoArray< T > & BiometricEvaluation::Memory::AutoArray< T >::operator= (AutoArray< T > *other*) throw Error::MemoryError)

Assignment operator overload performing a deep copy.

Parameters

in	*other*	AutoArray to be copied.

Returns

Reference to a new AutoArray object, the lvalue AutoArray.

Exceptions

Error::MemoryError	Could not allocate new memory.

Note

The signature for this operator overload is different than a traditional pass by constant reference to make use of the "copy-and-swap" idiom.

F.16.5 Member Data Documentation

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

Obtain an iterator to the beginning of the AutoArray.
Obtain an iterator to the end of the AutoArray.

Returns

Const iterator positioned at the first element of the AutoArray.
Iterator positioned at the one-past-last element of the AutoArray.

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

Obtain the number of accessible elements.

Returns

Number of accessible elements.

Note

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

F.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)

F.17.1 Member Typedef Documentation

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;
```

F.18 BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet Struct Reference

Public Member Functions

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

  Create a new CharacterSet struct.

Public Attributes

- uint16_t identifier
- string commonName
- string version

F.18.1 Constructor & Destructor Documentation

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.

F.18.2 Member Data Documentation

uint16_t BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::identifier

Identifier (000-999)

string BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::commonName

Common name of the character set

string BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::version

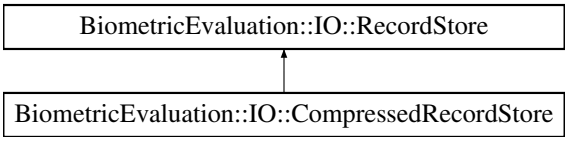
Optional version of the character set

F.19 BiometricEvaluation::IO::CompressedRecordStore Class Reference

Sibling-implemented RecordStore with Compression.

`#include <be_io_compressedrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::CompressedRecordStore:



Public Member Functions

- CompressedRecordStore (const string &name, const string &description, const string &recordStoreType, const string &parentDir, const string &compressorType) throw (Error::ObjectExists, Error::StrategyError)
- CompressedRecordStore (const string &name, const string &description, const string &recordStoreType, const string &parentDir, const Compressor::Kind &compressorType) throw (Error::ObjectExists, Error::StrategyError)
- CompressedRecordStore (const string &name, const string &parentDir, uint8_t mode=IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t const **throw** (Error::StrategyError)
- void const **throw** (Error::StrategyError)
- void insert (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
- void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t read (const string &key, void *const data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void replace (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)

- uint64_t length (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void flush (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t sequence (string &key, void *const data=NULL, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)

Static Public Attributes

- static const string BACKING_STORE
- static const string COMPRESSOR_TYPE_KEY

Additional Inherited Members

F.19.1 Detailed Description

Sibling-implemented RecordStore with Compression.

F.19.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const string & *name*, const string & *description*, const string & *recordStoreType*, const string & *parentDir*, const string & *compressorType*) throw Error::ObjectExists, Error::StrategyError)

Create a new CompressedRecordStore, read/write mode.
Parameters

in	*name*	The name of the store.
in	*description*	The store's description.
in	*recordStoreType*	The type of RecordStore subclass the internal RecordStores should be.
in	*parentDir*	The directory where the store is to be created.
in	*compressorType*	The type of compression that should be used within the internal RecordStores.

Exceptions

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

BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const string & *name*, const string & *description*, const string & *recordStoreType*, const string & *parentDir*, const Compressor::Kind & *compressorType*) throw Error::ObjectExists, Error::StrategyError)

Create a new CompressedRecordStore, read/write mode.
Parameters

in	*name*	The name of the store.
in	*description*	The store's description.
in	*recordStoreType*	The type of RecordStore subclass the internal RecordStores should be.
in	*parentDir*	The directory where the store is to be created.
in	*compressorType*	The type of compression that should be used within the internal RecordStores.

Exceptions

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

BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (const string & *name*, const string & *parentDir*, uint8_t *mode* = `IO::READWRITE`) throw Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing CompressedRecordStore.

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.

F.19.3 Member Function Documentation

void BiometricEvaluation::IO::CompressedRecordStore::insert (const string & *key*, const void *const *data*, const uint64_t *size*) throw Error::ObjectExists, Error::StrategyError) `[virtual]`

Insert a record into the store.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::CompressedRecordStore::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.

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

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

Parameters

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Replace a complete record in a store.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::CompressedRecordStore::length (const string & *key*) const 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.

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

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

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

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

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::CompressedRecordStore::setCursorAtKey (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.

void BiometricEvaluation::IO::CompressedRecordStore::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.

F.19.4 Member Data Documentation

const string BiometricEvaluation::IO::CompressedRecordStore::BACKING_STORE [static]

Name of the underlying store within this RS

const string BiometricEvaluation::IO::CompressedRecordStore::COMPRESSOR_TYPE_KEY [static]

Name of the key storing compressor type

F.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 }

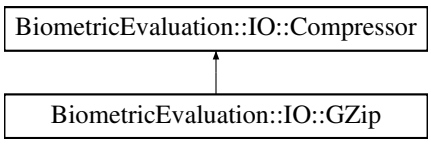
F.20.1 Detailed Description

Image compression algorithms.

F.21 BiometricEvaluation::IO::Compressor Class Reference

`#include <be_io_compressor.h>`

Inheritance diagram for BiometricEvaluation::IO::Compressor:



Public Types

- enum Kind { **GZIP** }

Public Member Functions

- Compressor ()

  Create a new Compressor object.
- virtual Memory::uint8Array compress (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize) const =0 throw (Error::StrategyError)

  Compress a buffer.
- virtual Memory::uint8Array compress (const Memory::uint8Array &uncompressedData) const =0 throw (Error::StrategyError)

  Compress a buffer.
- virtual void compress (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize, const string &outputFile) const =0 throw (Error::ObjectExists, Error::StrategyError)

  Compress a buffer.
- virtual void compress (const Memory::uint8Array &uncompressedData, const string &outputFile) const =0 throw (Error::ObjectExists, Error::StrategyError)

  Compress a buffer.
- virtual Memory::uint8Array compress (const string &inputFile) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Compress a file.
- virtual void compress (const string &inputFile, const string &outputFile) const =0 throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError)

  Compress a file.
- virtual Memory::uint8Array decompress (const uint8_t *const compressedData, uint64_t compressedDataSize) const =0 throw (Error::StrategyError)

  Decompress a compressed buffer.
- virtual Memory::uint8Array decompress (const Memory::uint8Array &compressedData) const =0 throw (Error::StrategyError)

  Decompress a compressed buffer.
- virtual Memory::uint8Array decompress (const string &inputFile) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Decompress a compressed buffer into a file.
- virtual void decompress (const Memory::uint8Array &compressedData, const string &outputFile) const =0 throw (Error::ObjectExists, Error::StrategyError)

  Decompress a file.
- virtual void decompress (const uint8_t *const compressedData, const uint64_t compressedDataSize, const string &outputFile) const =0 throw (Error::ObjectExists, Error::StrategyError)

  Decompress a file.
- virtual void decompress (const string &inputFile, const string &outputFile) const =0 throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError)

  Decompress a file.
- void setOption (const string &optionName, const string &optionValue) throw (Error::StrategyError)

  Assign a compressor option.
- void setOption (const string &optionName, int64_t optionValue) throw (Error::StrategyError)

Assign a compressor option.

- string getOption (const string &optionName) const throw (Error::ObjectDoesNotExist)

  Obtain a compressor option as an integer.
- int64_t getOptionAsInteger (const string &optionName) const throw (Error::ObjectDoesNotExist)

  Obtain a compressor option as an integer.
- void removeOption (const string &optionName) throw (Error::ObjectDoesNotExist)

  Remove a compressor option.
- virtual ~Compressor ()

Static Public Member Functions

- static string kindToString (Compressor::Kind compressor) throw (Error::ObjectDoesNotExist)

  Convert Kind enumeration to string.
- static Compressor::Kind stringToKind (const string &compressor) throw (Error::ObjectDoesNotExist)

  Convert string to Kind enumeration.
- static tr1::shared_ptr < Compressor > createCompressor (Compressor::Kind compressorKind=Compressor::GZIP) throw (Error::ObjectDoesNotExist)

Static Public Attributes

- static const string GZIPTYPE

F.21.1 Detailed Description

Implementations for compressing and decompressing data

F.21.2 Member Enumeration Documentation

enum BiometricEvaluation::IO::Compressor::Kind

Kinds of Compressors (for factory)

F.21.3 Constructor & Destructor Documentation

BiometricEvaluation::IO::Compressor::Compressor ( )

Create a new Compressor object.
Default compression options will be used.

virtual BiometricEvaluation::IO::Compressor::~Compressor ( ) `[virtual]`

Destructor

F.21.4 Member Function Documentation

static string BiometricEvaluation::IO::Compressor::kindToString (Compressor::Kind *compressor*) throw Error::ObjectDoesNotExist) `[static]`

Convert Kind enumeration to string.

Parameters

in	*compressor*	The Compressor to convert.

Returns

String representation of compressor.

Exceptions

Error::ObjectDoesNotExist	compressor is not a valid Compressor type.

static Compressor::Kind BiometricEvaluation::IO::Compressor::stringToKind (const string & *compressor*) throw Error::ObjectDoesNotExist) [static]

Convert string to Kind enumeration.

Parameters

in	*compressor*	The Compressor to convert.

Returns

Kind enumeration of compressor.

Exceptions

Error::ObjectDoesNotExist	compressor is not a valid Compressor type.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (const uint8_t *const *uncompressedData*, uint64_t *uncompressedDataSize*) const throw Error::StrategyError) [pure virtual]

Compress a buffer.

Parameters

uncompressedData	Uncompressed data buffer to compress.
uncompressedDataSize	Size of uncompressedData.

Returns

Compressed buffer.

Exceptions

Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (const Memory::uint8Array & *uncompressedData*) const throw Error::StrategyError) [pure virtual]

Compress a buffer.

Parameters

uncompressed-Data	Uncompressed data buffer to compress.

Returns

Compressed buffer.

Exceptions

Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::compress (const uint8_t *const *uncompressedData,* uint64_t *uncompressedDataSize,* const string & *outputFile*) const throw Error::ObjectExists, Error::StrategyError) `[pure virtual]`

Compress a buffer.
Parameters

uncompressed-Data	Uncompressed data buffer to compress.
uncompressed-DataSize	Size of uncompressedData.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::compress (const Memory::uint8Array & *uncompressedData,* const string & *outputFile*) const throw Error::ObjectExists, Error::StrategyError) `[pure virtual]`

Compress a buffer.
Parameters

uncompressed-Data	Uncompressed data buffer to compress.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (const string & *inputFile*) const throw Error::ObjectDoesNotExist, Error::StrategyError) `[pure virtual]`

Compress a file.

Parameters

inputFile	Path to file to compress.

Returns

Compressed buffer.

Exceptions

Error::ObjectDoesNotExist	Input file does not exist.
Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::compress (const string & *inputFile*, const string & *outputFile*) const throw Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError) [pure virtual]

Compress a file.

Parameters

inputFile	Path to file to compress.
outputFile	Path to location where compressed version will be saved.

Exceptions

Error::ObjectDoesNotExist	Input file does not exist.
Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (const uint8_t *const *compressedData*, uint64_t *compressedDataSize*) const throw Error::StrategyError) [pure virtual]

Decompress a compressed buffer.

Parameters

compressedData	Compressed data buffer to decompress.
compressedDataSize	Size of compressedData.

Returns

Decompressed buffer.

Exceptions

Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (const Memory::uint8Array & *compressedData*) const throw Error::StrategyError) [pure virtual]

Decompress a compressed buffer.

Parameters

compressed-Data	Compressed data buffer to decompress.

Returns

Decompressed buffer.

Exceptions

Error::StrategyError	Error in decompression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (const string & *inputFile*) const throw Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

Decompress a compressed buffer into a file.
Parameters

inputFile	Location to save compressed file.

Returns

Decompressed buffer.

Exceptions

Error::StrategyError	Error in decompression unit.
Error::ObjectDoesNot-Exists	Output file already exists.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::decompress (const Memory::uint8Array & *compressedData*, const string & *outputFile*) const throw Error::ObjectExists, Error::StrategyError) [pure virtual]

Decompress a file.
Parameters

compressed-Data	Compressed data buffer to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::decompress (const uint8_t *const *compressedData*, const uint64_t *compressedDataSize*, const string & *outputFile*) const throw Error::ObjectExists, Error::StrategyError) [pure virtual]

Decompress a file.

Parameters

compressed-Data	Compressed data buffer to decompress.
compressed-DataSize	Size of compressedData.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

virtual void BiometricEvaluation::IO::Compressor::decompress (const string & *inputFile*, const string & *outputFile*) const throw Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError) `[pure virtual]`

Decompress a file.

Parameters

inputFile	Path to file to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectDoesNot-Exist	Input file does not exist.
Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implemented in BiometricEvaluation::IO::GZip.

void BiometricEvaluation::IO::Compressor::setOption (const string & *optionName*, const string & *optionValue*) throw Error::StrategyError)

Assign a compressor option.

Will overwrite existing values without warning.

Parameters

optionName	Name of the option to add.
optionValue	Value of the option.

Exceptions

Error::StrategyError	Error setting option.

void BiometricEvaluation::IO::Compressor::setOption (const string & *optionName*, int64_t *optionValue*) throw Error::StrategyError)

Assign a compressor option.

Will overwrite existing values without warning.

Parameters

optionName	Name of the option to add.
optionValue	Value of the option.

Exceptions

Error::StrategyError	Error setting option.

string BiometricEvaluation::IO::Compressor::getOption (const string & *optionName*) const throw Error::ObjectDoesNotExist)

Obtain a compressor option as an integer.
Parameters

optionName	Name of the option to obtain.

Returns

Value of compressor option.

int64_t BiometricEvaluation::IO::Compressor::getOptionAsInteger (const string & *optionName*) const throw Error::ObjectDoesNotExist)

Obtain a compressor option as an integer.
Parameters

optionName	Name of the option to obtain.

Returns

Value of compressor option.

Exceptions

Error::ObjectDoesNotExist	The option was never set.

void BiometricEvaluation::IO::Compressor::removeOption (const string & *optionName*) throw Error::ObjectDoesNotExist)

Remove a compressor option.
Parameters

optionName	Name of the option to remove.

static tr1::shared_ptr<Compressor> BiometricEvaluation::IO::Compressor::createCompressor (Compressor::Kind *compressorKind* = `Compressor::GZIP`) throw Error::ObjectDoesNotExist) `[static]`

Compressor factory.

Parameters

compressorKind	A known kind of compressor.

Returns

A new compressor with default options.

Exceptions

Error::ObjectDoesNotExist	Invalid compressor type.

F.21.5 Member Data Documentation

const string BiometricEvaluation::IO::Compressor::GZIPTYPE [static]

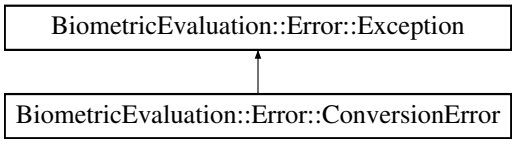
String representations of the compressors

F.22 BiometricEvaluation::Error::ConversionError Class Reference

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

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ConversionError:



Public Member Functions

- ConversionError ()
- ConversionError (string info)

F.22.1 Detailed Description

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

F.22.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ConversionError::ConversionError ( )

Construct a ConversionError object with the default information string.

BiometricEvaluation::Error::ConversionError::ConversionError (string *info*)

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

F.23 BiometricEvaluation::Image::Coordinate Struct Reference

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

`#include <be_image.h>`

Public Member Functions

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

  Create a Coordinate struct.

Public Attributes

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

F.23.1 Detailed Description

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

F.23.2 Constructor & Destructor Documentation

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

Create a Coordinate struct.

Parameters

in	*x*	X-coordinate
in	*y*	Y-coordinate
in	*xDistance*	X-coordinate distance from origin
in	*yDistance*	Y-coordinate distance from origin

F.23.3 Member Data Documentation

uint32_t BiometricEvaluation::Image::Coordinate::x

X-coordinate

uint32_t BiometricEvaluation::Image::Coordinate::y

Y-coordinate

float BiometricEvaluation::Image::Coordinate::xDistance

X-coordinate distance from origin

float BiometricEvaluation::Image::Coordinate::yDistance

Y-coordinate distance from origin

F.24 BiometricEvaluation::Feature::CorePoint Struct Reference

Representation of the core.

`#include <be_feature_minutiae.h>`

Public Member Functions

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

  Create a CorePoint struct.

Public Attributes

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

F.24.1 Detailed Description

Representation of the core.

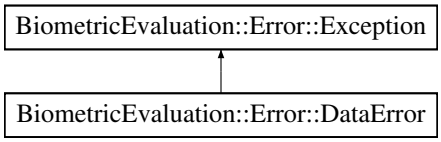
A core has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

F.25 BiometricEvaluation::Error::DataError Class Reference

Error when reading data from an external source.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::DataError:



Public Member Functions

- DataError ()
- DataError (string info)

F.25.1 Detailed Description

Error when reading data from an external source.

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

F.25.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::DataError::DataError ( )

Construct a DataError object with the default information string.

BiometricEvaluation::Error::DataError::DataError (string *info*)

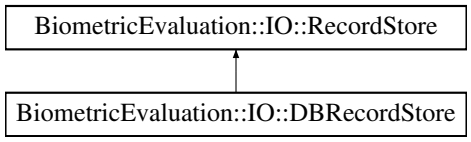
Construct a DataError object with an information string appended to the default information string.

F.26 BiometricEvaluation::IO::DBRecordStore Class Reference

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

`#include <be_io_dbrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::DBRecordStore:



Public Member Functions

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

  Obtain real storage utilization.
- void sync () const throw (Error::StrategyError)
- void insert (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
- void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t read (const string &key, void *const data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void replace (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t length (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void flush (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t sequence (string &key, void *const data=NULL, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)

Additional Inherited Members

F.26.1 Detailed Description

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

F.26.2 Constructor & Destructor Documentation

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.

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.

F.26.3 Member Function Documentation

uint64_t BiometricEvaluation::IO::DBRecordStore::getSpaceUsed ( ) const throw Error::StrategyError) [virtual]

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::DBRecordStore::sync ( ) const throw Error::StrategyError) [virtual]

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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 inserted.
in	*data*	The data for the record.
in	*size*	The size, in bytes, of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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.

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

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

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::DBRecordStore::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.

uint64_t BiometricEvaluation::IO::DBRecordStore::length (const string & *key*) const 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.

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

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

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

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

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::DBRecordStore::setCursorAtKey (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.

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.

F.27 BiometricEvaluation::Feature::DeltaPoint Struct Reference

Representation of the delta.

`#include <be_feature_minutiae.h>`

Public Member Functions

- DeltaPoint (Image::Coordinate coordinate, bool has_angle=false, int angle1=0, int angle2=0, int angle3=0)

  Create a DeltaPoint struct.

Public Attributes

- Image::Coordinate **coordinate**
- bool **has_angle**
- int **angle1**
- int **angle2**
- int **angle3**

F.27.1 Detailed Description

Representation of the delta.

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

F.28 BiometricEvaluation::View::AN2KView::DeviceMonitoringMode Class Reference

The level of human monitoring for the image capture device.

`#include <be_view_an2kview.h>`

Public Types

- enum Kind {
  Controlled, Assisted, Observed, Unattended,
  Unknown, NA }

F.28.1 Detailed Description

The level of human monitoring for the image capture device.

F.28.2 Member Enumeration Documentation

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

F.29 BiometricEvaluation::DataInterchange::AN2KRecord::DomainName Struct Reference

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

`#include <be_data_interchange_an2k.h>`

Public Member Functions

- DomainName (string identifier="", string version="")

  Create a DomainName struct.

Public Attributes

- string identifier
- string version

F.29.1 Detailed Description

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

F.29.2 Constructor & Destructor Documentation

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.

F.29.3 Member Data Documentation

string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::identifier

Unique identifier for agency, entity, or implementation.

string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::version

Optional version of the implementation

F.30 BiometricEvaluation::Feature::AN2K7Minutiae::EncodingMethod Class Reference

Methods for encoding minutiae data in an AN2K record.

`#include <be_feature_an2k7minutiae.h>`

Public Types

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

F.30.1 Detailed Description

Methods for encoding minutiae data in an AN2K record.

F.31 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification-::Entry Struct Reference

Public Member Functions

- Entry (bool standard, std::string code)

Public Attributes

- bool standard
- std::string code

F.31.1 Constructor & Destructor Documentation

BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::Entry (bool *standard*, std::string *code*)

Create an Entry struct.

Parameters

standard	Whether or not code is a standard AN2K pattern classification code.
code	AN2K or user-defined pattern classification code.

F.31.2 Member Data Documentation

bool BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::standard

Whether code is a standard AN2K pattern classification code.

std::string BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::code

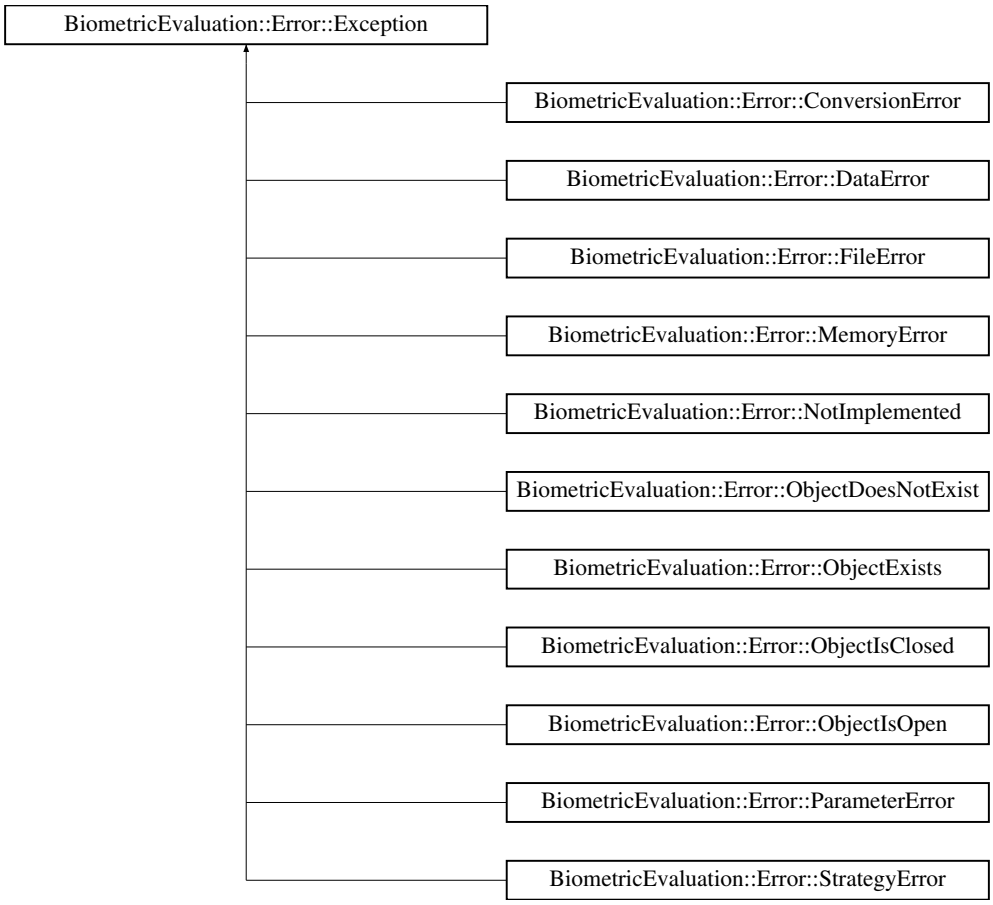
AN2K or user-defined pattern classification code.

F.32 BiometricEvaluation::Error::Exception Class Reference

The parent class of all BiometricEvaluation exceptions.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::Exception:



Public Member Functions

- Exception ()
- Exception (string info)
- string getInfo ()

F.32.1 Detailed Description

The parent class of all BiometricEvaluation exceptions.

The classes derived from this class will have a default information string set indicating the type of exception. Any additional information string is appended to that string.

F.32.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::Exception::Exception ( )

Construct an Exception object without an information string.

BiometricEvaluation::Error::Exception::Exception (string *info*)

Construct an Exception object with an information string.

Parameters

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

F.32.3 Member Function Documentation

string BiometricEvaluation::Error::Exception::getInfo ( )

Obtain the information string associated with the exception.

Returns

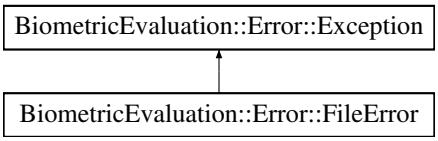
The information string.

F.33 BiometricEvaluation::Error::FileError Class Reference

File error when opening, reading, writing, etc.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::FileError:



Public Member Functions

- FileError ()
- FileError (string info)

F.33.1 Detailed Description

File error when opening, reading, writing, etc.

F.33.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::FileError::FileError ( )

Construct a FileError object with the default information string.

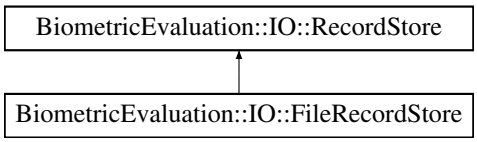
BiometricEvaluation::Error::FileError::FileError (string *info*)

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

F.34 BiometricEvaluation::IO::FileRecordStore Class Reference

`#include <be_io_filerecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::FileRecordStore:



Public Member Functions

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

  Obtain real storage utilization.
- void insert (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
- void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t read (const string &key, void *const data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void replace (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t length (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void flush (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t sequence (string &key, void *const data=NULL, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)

Protected Member Functions

- string **canonicalName** (const string &name) const

Additional Inherited Members

F.34.1 Detailed Description

Class to represent the record store data storage mechanism implemented as files for each record.

Note

For the methods that take a key parameter, Error::StrategyError will be thrown if the key string is not compliant. A FileRecordStore has the additional requirement that a key name may not contain path delimiter characters ('/' and '\'), or begin with whitespace.

F.34.2 Constructor & Destructor Documentation

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.

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.

F.34.3 Member Function Documentation

uint64_t BiometricEvaluation::IO::FileRecordStore::getSpaceUsed ( ) const throw Error::StrategyError) [virtual]

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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 inserted.
in	*data*	The data for the record.
in	*size*	The size, in bytes, of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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.

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

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

in	*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.

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.

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

Return the length of a record.
Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

Commit the record's data to storage.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

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

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

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::FileRecordStore::setCursorAtKey (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.

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.

F.35 BiometricEvaluation::Finger::FingerImageCode Class Reference

`#include <be_finger.h>`

Public Types

- enum **Kind** {
  EJI = 0, **RolledTip**, **FullFingerRolled**, **FullFingerPlainLeft**,
  FullFingerPlainCenter, **FullFingerPlainRight**, **ProximalSegment**, **DistalSegment**,
  MedialSegment, **NA** }

F.35.1 Detailed Description

Joint and tip codes.

F.36 BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem Struct Reference

Representation of information about a fingerprint reader system.

`#include <be_feature_an2k7minutiae.h>`

Public Attributes

- string name
- EncodingMethod::Kind method
- string equipment

F.36.1 Detailed Description

Representation of information about a fingerprint reader system.

F.36.2 Member Data Documentation

string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::name

Name for system that encoded minutiae

EncodingMethod::Kind BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::method

Method used to encoded minutiae

string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::equipment

Optional ID for equipment used in system

F.37 BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition Struct Reference

Locations of an individual finger segment in a slap.

`#include <be_finger_an2kview_capture.h>`

Public Member Functions

- FingerSegmentPosition (const Finger::Position::Kind fingerPosition, const Image::CoordinateSet coordinates)

  Create an FingerSegmentPosition struct.

Public Attributes

- Finger::Position::Kind fingerPosition
- Image::CoordinateSet coordinates

F.37.1 Detailed Description

Locations of an individual finger segment in a slap.

F.37.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::FingerSegmentPosition (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.

F.37.3 Member Data Documentation

Finger::Position::Kind BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition-::fingerPosition

Finger depicted in this segment

Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition-::coordinates

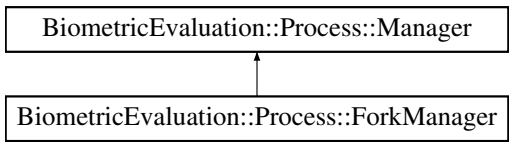
Points composing the segmented polygon

F.38 BiometricEvaluation::Process::ForkManager Class Reference

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

`#include <be_process_forkmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::ForkManager:



Public Member Functions

- ForkManager ()
- tr1::shared_ptr< WorkerController > addWorker (tr1::shared_ptr< Worker > worker)

  Adds a Worker to be managed by this Manager.
- void startWorkers (bool wait=true, bool communicate=false) throw (Error::ObjectExists, Error::StrategyError)

  Begin Worker's work.
- void startWorker (tr1::shared_ptr< WorkerController > worker, bool wait=true, bool communicate=false) throw (Error::ObjectExists, Error::StrategyError)

  Start a worker.
- int32_t stopWorker (tr1::shared_ptr< WorkerController > workerController) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Ask Worker to exit.
- void broadcastSignal (int signo)

  Send a POSIX signal to all workers.
- bool responsibleFor (const pid_t pid) const

  Obtain whether or not this ForkManager is responsbile for a particular PID.
- void setNotWorking (const pid_t pid)

  Set Status.isWorking for PID to false.
- void markAllFinished ()

  Call setNotWorking() for all PIDs known to this ForkManager.
- bool getIsWorkingStatus (const pid_t pid) const

  Get Status.isWorking for PID.
- ~ForkManager ()

  ForkManager destructor.

Static Public Attributes

- static std::list< ForkManager * > FORKMANAGERS

  List of all instantiated ForkManagers.

Additional Inherited Members

F.38.1 Detailed Description

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

F.38.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::ForkManager::ForkManager ( )

ForkManager constructor.

F.38.3 Member Function Documentation

tr1::shared_ptr<WorkerController> BiometricEvaluation::Process::ForkManager::addWorker (tr1::shared_ptr< Worker > *worker*) `[virtual]`

Adds a Worker to be managed by this Manager.

Parameters

worker	A Worker instance to run.

Returns

shared_ptr to worker.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::ForkManager::startWorkers (bool *wait* = `true`, bool *communicate* = `false`) throw Error::ObjectExists, Error::StrategyError) `[virtual]`

Begin Worker's work.

Parameters

in	*wait*	Whether or not to wait for all Workers to return before returning.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	At least one Worker is already working.
Error::StrategyError	Problem forking.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::ForkManager::startWorker (tr1::shared_ptr< WorkerController > *worker*, bool *wait* = `true`, bool *communicate* = `false`) throw Error::ObjectExists, Error::StrategyError) `[virtual]`

Start a worker.

Parameters

	worker	Pointer to a WorkerController that is being managed by this Manager instance.
	wait	Whether or not to wait for this Worker to exit before returning control to the caller.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

Implements BiometricEvaluation::Process::Manager.

int32_t BiometricEvaluation::Process::ForkManager::stopWorker (tr1::shared_ptr< WorkerController > *workerController*) throw Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Ask Worker to exit.

Sends SIGUSR1 to the Worker, which ForkManager will handle automatically.

Parameters

worker-Controller	Pointer to the ForkWorkerController that should be stopped.

Returns

Exit status of worker.

Exceptions

Error::ObjectDoesNot-Exist	worker is not working.
Error::StrategyError	Problem sending the signal.

Attention

Do not call stopWorker() when communication is enabled unless you will be finished with communication for all Workers at that point. This creates a race condition for reads()/writes() when the Worker exits.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::ForkManager::broadcastSignal (int *signo*)

Send a POSIX signal to all workers.

Parameters

in	*signo*	The signal to send.

bool BiometricEvaluation::Process::ForkManager::responsibleFor (const pid_t *pid*) const

Obtain whether or not this ForkManager is responsbile for a particular PID.

Parameters

in	*pid*	PID in question

Returns

true if this ForkManager spawned pid, false otherwise.

void BiometricEvaluation::Process::ForkManager::setNotWorking (const pid_t *pid*)

Set Status.isWorking for PID to false.

Parameters

in	*pid*	PID whose inWorking flag should be set to false

Exceptions

Error::ObjectDoesNot-Exist	PID not under this manager's control.

bool BiometricEvaluation::Process::ForkManager::getIsWorkingStatus (const pid_t *pid*) const

Get Status.isWorking for PID.

Parameters

in	*pid*	PID whose inWorking flag should be queried

Exceptions

Error::ObjectDoesNot-Exist	PID not under this manager's control.

F.38.4 Member Data Documentation

std::list<ForkManager*> BiometricEvaluation::Process::ForkManager::FORKMANAGERS [static]

List of all instantiated ForkManagers.

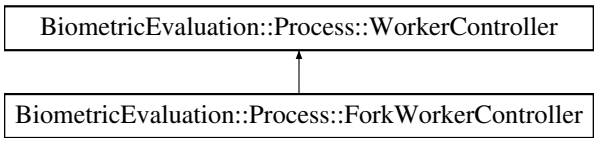
This is not a list of managed pointers to ForkManagers. If it was, the smart pointer's destructor would attempt to delete the object being pointed to at program termination, which is ultimately sometime after the destructor of the ForkManager itself was called.

F.39 BiometricEvaluation::Process::ForkWorkerController Class Reference

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

`#include <be_process_forkmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::ForkWorkerController:



Public Member Functions

- void throw (Error::ObjectExists)

  Reuse the Worker.
- ~ForkWorkerController ()

  ForkWorkerController destructor.

Static Public Member Functions

- static void _stop (int signal)

  Tell _staticWorker to stop.

Public Attributes

- bool const

  Obtain whether or not Worker is working.
- pid_t const

  Obtain the PID of this process this instance represents.

Friends

- void ForkManager::startWorkers (bool wait, bool communicate) throw (Error::ObjectExists, Error::StrategyError)

  Begin Worker's work.
- void ForkManager::startWorker (tr1::shared_ptr< WorkerController > worker, bool wait, bool communicate) throw (Error::ObjectExists, Error::StrategyError)

  Restart a completed Worker.
- int32_t ForkManager::stopWorker (tr1::shared_ptr< WorkerController > workerController) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Ask Worker to exit.
- tr1::shared_ptr< WorkerController > ForkManager::addWorker (tr1::shared_ptr< Worker > worker)

  Adds a Worker to be managed by this Manager.

Additional Inherited Members

F.39.1 Detailed Description

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

F.39.2 Member Function Documentation

void BiometricEvaluation::Process::ForkWorkerController::throw (Error::ObjectExists) `[virtual]`

Reuse the Worker.
Exceptions

Error::ObjectExists	The previously started Worker is still running.

Reimplemented from BiometricEvaluation::Process::WorkerController.

static void BiometricEvaluation::Process::ForkWorkerController::_stop (int *signal*) `[static]`

Tell _staticWorker to stop.
Called by the child process instance when SIGUSR1 is received.
Parameters

signal	The signal caught that prompted this function to be called (SIGUSR1).

F.39.3 Friends And Related Function Documentation

void ForkManager::startWorkers (bool *wait*, bool *communicate*) throw Error::ObjectExists, Error::StrategyError) `[friend]`

Begin Worker's work.
Parameters

in	*wait*	Whether or not to wait for all Workers to return before returning.

in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	One or more of the Workers is already working.
Error::StrategyError	Problem forking.

void ForkManager::startWorker (tr1::shared_ptr< WorkerController > *worker*, bool *wait*, bool *communicate*) throw Error::ObjectExists, Error::StrategyError) [friend]

Restart a completed Worker.

Parameters

	worker	Pointer to a WorkerController that is being managed by this Manager instance.
	wait	Whether or not to wait for this Worker to exit before returning control to the caller.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

int32_t ForkManager::stopWorker (tr1::shared_ptr< WorkerController > *workerController*) throw Error::ObjectDoesNotExist, Error::StrategyError) [friend]

Ask Worker to exit.

Sends SIGUSR1 to the Worker, which ForkManager will handle automatically.

Parameters

workerController	Pointer to the ForkWorkerController that should be stopped.

Returns

Exit status of worker.

Exceptions

Error::ObjectDoesNotExist	worker is not working.
Error::StrategyError	Problem sending the signal.

tr1::shared_ptr<WorkerController> ForkManager::addWorker (tr1::shared_ptr< Worker > *worker*) [friend]

Adds a Worker to be managed by this Manager.

Parameters

worker	A Worker instance to run.

Returns

shared_ptr to worker.

F.39.4 Member Data Documentation

bool BiometricEvaluation::Process::ForkWorkerController::const

Obtain whether or not Worker is working.

Returns

Whether or not the Worker is working.

pid_t BiometricEvaluation::Process::ForkWorkerController::const

Obtain the PID of this process this instance represents.

Returns

pid of the process this instance represents.

Note

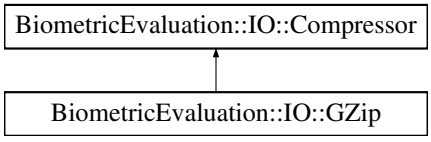
Call isRunning() before doing anything with the PID returned from this function.

F.40 BiometricEvaluation::IO::GZip Class Reference

Compressor for gzip compression from zlib.

`#include <be_io_gzip.h>`

Inheritance diagram for BiometricEvaluation::IO::GZip:



Public Member Functions

- Memory::uint8Array compress (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize) const throw (Error::StrategyError)

  Compress a buffer.
- Memory::uint8Array compress (const Memory::uint8Array &uncompressedData) const throw (Error::StrategyError)

  Compress a buffer.
- void compress (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize, const string &outputFile) const throw (Error::ObjectExists, Error::StrategyError)

Compress a buffer.

- void compress (const Memory::uint8Array &uncompressedData, const string &outputFile) const throw (Error::ObjectExists, Error::StrategyError)

  Compress a buffer.
- Memory::uint8Array compress (const string &inputFile) const throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Compress a file.
- void compress (const string &inputFile, const string &outputFile) const throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError)

  Compress a file.
- Memory::uint8Array decompress (const uint8_t *const compressedData, uint64_t compressedDataSize) const throw (Error::StrategyError)

  Decompress a compressed buffer.
- Memory::uint8Array decompress (const Memory::uint8Array &compressedData) const throw (Error::StrategyError)

  Decompress a compressed buffer.
- Memory::uint8Array decompress (const string &input) const throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Decompress a compressed buffer into a file.
- void decompress (const string &inputFile, const string &outputFile) const throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError)

  Decompress a file.
- void decompress (const uint8_t *const compressedData, const uint64_t compressedDataSize, const string &outputFile) const throw (Error::ObjectExists, Error::StrategyError)

  Decompress a file.
- void decompress (const Memory::uint8Array &compressedData, const string &outputFile) const throw (Error::ObjectExists, Error::StrategyError)

  Decompress a file.

Static Public Attributes

- static const string COMPRESSION_LEVEL
- static const string COMPRESSION_STRATEGY
- static const string COMPRESSION_METHOD
- static const string INPUT_DATA_TYPE
- static const string WINDOW_BITS
- static const string MEMORY_LEVEL
- static const string CHUNK_SIZE

Additional Inherited Members

F.40.1 Detailed Description

Compressor for gzip compression from zlib.

F.40.2 Member Function Documentation

Memory::uint8Array BiometricEvaluation::IO::GZip::compress (const uint8_t *const *uncompressedData,* uint64_t *uncompressedDataSize*) const throw Error::StrategyError) `[virtual]`

Compress a buffer.

Parameters

uncompressed-Data	Uncompressed data buffer to compress.
uncompressed-DataSize	Size of uncompressedData.

Returns

Compressed buffer.

Exceptions

Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::compress (const Memory::uint8Array & *uncompressedData*) const throw Error::StrategyError) `[virtual]`

Compress a buffer.

Parameters

uncompressed-Data	Uncompressed data buffer to compress.

Returns

Compressed buffer.

Exceptions

Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::compress (const uint8_t *const *uncompressedData,* uint64_t *uncompressedDataSize,* const string & *outputFile*) const throw Error::ObjectExists, Error::StrategyError) `[virtual]`

Compress a buffer.

Parameters

uncompressed-Data	Uncompressed data buffer to compress.
uncompressed-DataSize	Size of uncompressedData.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::compress (const Memory::uint8Array & *uncompressedData,* const string & *outputFile*) const throw Error::ObjectExists, Error::StrategyError) `[virtual]`

Compress a buffer.

Parameters

uncompressed-Data	Uncompressed data buffer to compress.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::compress (const string & *inputFile*) const throw Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Compress a file.
Parameters

inputFile	Path to file to compress.

Returns

Compressed buffer.

Exceptions

Error::ObjectDoesNot-Exist	Input file does not exist.
Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::compress (const string & *inputFile*, const string & *outputFile*) const throw Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError) `[virtual]`

Compress a file.
Parameters

inputFile	Path to file to compress.
outputFile	Path to location where compressed version will be saved.

Exceptions

Error::ObjectDoesNot-Exist	Input file does not exist.
Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (const uint8_t *const *compressedData*, uint64_t *compressedDataSize*) const throw Error::StrategyError) `[virtual]`

Decompress a compressed buffer.

Parameters

compressed-Data	Compressed data buffer to decompress.
compressed-DataSize	Size of compressedData.

Returns

Decompressed buffer.

Exceptions

Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (const Memory::uint8Array & *compressedData*) const throw Error::StrategyError) `[virtual]`

Decompress a compressed buffer.

Parameters

compressed-Data	Compressed data buffer to decompress.

Returns

Decompressed buffer.

Exceptions

Error::StrategyError	Error in decompression unit.

Implements BiometricEvaluation::IO::Compressor.

Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (const string & *inputFile*) const throw Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Decompress a compressed buffer into a file.

Parameters

inputFile	Location to save compressed file.

Returns

Decompressed buffer.

Exceptions

Error::StrategyError	Error in decompression unit.
Error::ObjectDoesNot-Exists	Output file already exists.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::decompress (const string & *inputFile*, const string & *outputFile*) const throw Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError) `[virtual]`

Decompress a file.

Parameters

inputFile	Path to file to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectDoesNotExist	Input file does not exist.
Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::decompress (const uint8_t *const *compressedData,* const uint64_t *compressedDataSize,* const string & *outputFile*) const throw Error::ObjectExists, Error::StrategyError) `[virtual]`

Decompress a file.

Parameters

compressedData	Compressed data buffer to decompress.
compressedDataSize	Size of compressedData.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

void BiometricEvaluation::IO::GZip::decompress (const Memory::uint8Array & *compressedData,* const string & *outputFile*) const throw Error::ObjectExists, Error::StrategyError) `[virtual]`

Decompress a file.

Parameters

compressedData	Compressed data buffer to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists	Output file already exists.
Error::StrategyError	Error in compression unit.

Implements BiometricEvaluation::IO::Compressor.

F.40.3 Member Data Documentation

const string BiometricEvaluation::IO::GZip::COMPRESSION_LEVEL `[static]`

How thorough the compression should be

const string BiometricEvaluation::IO::GZip::COMPRESSION_STRATEGY [static]

Which underlying algorithm to use

const string BiometricEvaluation::IO::GZip::COMPRESSION_METHOD [static]

Which underlying method in the compressor

const string BiometricEvaluation::IO::GZip::INPUT_DATA_TYPE [static]

The type of data being compressed

const string BiometricEvaluation::IO::GZip::WINDOW_BITS [static]

Window size

const string BiometricEvaluation::IO::GZip::MEMORY_LEVEL [static]

How much memory for internal compression state

const string BiometricEvaluation::IO::GZip::CHUNK_SIZE [static]

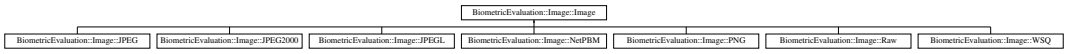
How many bytes to work at a time

F.41 BiometricEvaluation::Image::Image Class Reference

Represent attributes common to all images.

`#include <be_image_image.h>`

Inheritance diagram for BiometricEvaluation::Image::Image:



Public Member Functions

- Image (const uint8_t *data, const uint64_t size, const Size dimensions, const uint32_t depth, const Resolution resolution, const CompressionAlgorithm::Kind compression) throw (Error::DataError, Error::StrategyError)

  Parent constructor for all Image classes.

- Image (const uint8_t *data, const uint64_t size, const CompressionAlgorithm::Kind compression) throw (Error::DataError, Error::StrategyError)

  Parent constructor for all Image classes.

- virtual Memory::AutoArray < uint8_t > const throw (Error::DataError)=0

  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.

Static Public Member Functions

- static uint64_t valueInColorspace (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.
- static tr1::shared_ptr< Image > openImage (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)

  Determine the image type of a buffer of image data and create an Image object.
- static tr1::shared_ptr< Image > openImage (const Memory::uint8Array &data) throw (Error::DataError, Error::StrategyError)

  Determine the image type of a buffer of image data and create an Image object.
- static tr1::shared_ptr< Image > openImage (const string &path) throw (Error::DataError, Error::ObjectDoesNotExist, Error::StrategyError)

  Determine the image type of an image file and create an Image object.
- static CompressionAlgorithm::Kind getCompressionAlgorithm (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.
- static CompressionAlgorithm::Kind getCompressionAlgorithm (const Memory::uint8Array &data)

  Determine the compression algorithm of a buffer of image data.
- static CompressionAlgorithm::Kind getCompressionAlgorithm (const string &path) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Determine the compression algorithm of a file.

Public Attributes

- CompressionAlgorithm::Kind const

  Accessor for the CompressionAlgorithm of the image.
- Resolution const

  Accessor for the resolution of the image.
- Memory::AutoArray< uint8_t > const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- Size const

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

  Accessor for the color depth of the image in bits.

Static Public Attributes

- static const uint32_t bitsPerComponent = 8

Protected Member Functions

- void setResolution (const Resolution resolution)

  Mutator for the resolution of the image .
- void setDimensions (const Size dimensions)

  Mutator for the dimensions of the image in pixels.
- void setDepth (const uint32_t depth)

  Mutator for the color depth of the image in bits.

Protected Attributes

- Memory::AutoArray< uint8_t > _raw_data

F.41.1 Detailed Description

Represent attributes common to all images.

Images are represented by their size, depth, and resolution on the X and Y axes. The image data can be of any format, raw, JPEG, etc. Implementations of this abstraction provide the getRawData() method to convert image data to 'raw' format.

Image resolution is in pixels per centimeter, and the coordinate system has the origin at the upper left of the image.

F.41.2 Constructor & Destructor Documentation

BiometricEvaluation::Image::Image::Image (const uint8_t * *data*, const uint64_t *size*, const Size *dimensions*, const uint32_t *depth*, const Resolution *resolution*, const CompressionAlgorithm::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.

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.

F.41.3 Member Function Documentation

virtual Memory::AutoArray<uint8_t> const BiometricEvaluation::Image::Image::throw (Error::DataError) [pure virtual]

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

Returns

Raw image data.

Exceptions

Error::DataError	Error decompressing image data.

Implemented in BiometricEvaluation::Image::NetPBM, BiometricEvaluation::Image::JPEG, BiometricEvaluation::Image::JPEG2000, BiometricEvaluation::Image::PNG, BiometricEvaluation::Image::JPEGL, BiometricEvaluation::Image::Raw, and BiometricEvaluation::Image::WSQ.

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::NetPBM, BiometricEvaluation::Image::JPEG2000, BiometricEvaluation::Image::PNG, BiometricEvaluation::Image::JPEG, BiometricEvaluation::Image::Raw, BiometricEvaluation::Image::WSQ, and BiometricEvaluation::Image::JPEGL.

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.

static tr1::shared_ptr<Image> BiometricEvaluation::Image::Image::openImage (const uint8_t * *data*, const uint64_t *size*) throw Error::DataError, Error::StrategyError) `[static]`

Determine the image type of a buffer of image data and create an Image object.

Parameters

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

Returns

Image representation of the input data buffer.

Exceptions

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

static tr1::shared_ptr<Image> BiometricEvaluation::Image::Image::openImage (const Memory::uint8Array & *data*) throw Error::DataError, Error::StrategyError) [static]

Determine the image type of a buffer of image data and create an Image object.

Parameters

in	*data*	The image data.

Returns

Image representation of the input data buffer.

Exceptions

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

static tr1::shared_ptr<Image> BiometricEvaluation::Image::Image::openImage (const string & *path*) throw Error::DataError, Error::ObjectDoesNotExist, Error::StrategyError) [static]

Determine the image type of an image file and create an Image object.

Parameters

in	*path*	Path to image data.

Returns

Image representation of the input data buffer.

Exceptions

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

static CompressionAlgorithm::Kind BiometricEvaluation::Image::Image::getCompressionAlgorithm (const uint8_t * *data*, const uint64_t *size*) [static]

Determine the compression algorithm of a buffer of image data.

Parameters

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

Returns

Compression algorithm used in the buffer.

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation Framework is found.

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

Determine the compression algorithm of a buffer of image data.

Parameters

in	*data*	The image data.

Returns

Compression algorithm used in the buffer.

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation Framework is found.

static CompressionAlgorithm::Kind BiometricEvaluation::Image::Image::getCompressionAlgorithm (const string & *path*) throw Error::ObjectDoesNotExist, Error::StrategyError) [static]

Determine the compression algorithm of a file.

Parameters

in	*path*	Path to file.

Returns

Compression algorithm used in the file.

Exceptions

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

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation Framework is found.

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

Mutator for the resolution of the image .

Parameters

in	*resolution*	Resolution struct.

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

Mutator for the dimensions of the image in pixels.
Parameters

in	*dimensions*	Dimensions of image (pixel).

void BiometricEvaluation::Image::Image::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).

F.41.4 Member Data Documentation

CompressionAlgorithm::Kind BiometricEvaluation::Image::Image::const

Accessor for the CompressionAlgorithm of the image.

Returns

Type of compression used on the data that will be returned from getData().

Resolution BiometricEvaluation::Image::Image::const

Accessor for the resolution of the image.

Returns

Resolution struct

Memory::AutoArray<uint8_t> BiometricEvaluation::Image::Image::const

Accessor for the image data. The data returned is likely encoded in a specialized format.

Returns

Image data.

Size BiometricEvaluation::Image::Image::const

Accessor for the dimensions of the image in pixels.

Returns

Coordinate object containing dimensions in pixels.

uint32_t BiometricEvaluation::Image::Image::const

Accessor for the color depth of the image in bits.

Returns

The color depth of the image (bit).

const uint32_t BiometricEvaluation::Image::Image::bitsPerComponent = 8 [static]

Number of bits per color component

Memory::AutoArray<uint8_t> BiometricEvaluation::Image::Image::_raw_data [mutable], [protected]

Raw image data, populated on demand

F.42 BiometricEvaluation::Finger::Impression Class Reference

Finger and palm impression types.

`#include <be_finger.h>`

Public Types

- enum **Kind** {
  LiveScanPlain = 0, **LiveScanRolled**, **NonLiveScanPlain**, **NonLiveScanRolled**,
  LatentImpression, **LatentTracing**, **LatentPhoto**, **LatentLift**,
  LiveScanVerticalSwipe, **LiveScanPalm**, **NonLiveScanPalm**, **LatentPalmImpression**,
  LatentPalmTracing, **LatentPalmPhoto**, **LatentPalmLift**, **LiveScanOpticalContactPlain**,
  LiveScanOpticalContactRolled, **LiveScanNonOpticalContactPlain**, **LiveScanNonOpticalContact-Rolled**, **LiveScanOpticalContactlessPlain**,
  LiveScanOpticalContactlessRolled, **LiveScanNonOpticalContactlessPlain**, **LiveScanNonOpticalContactless-Rolled**, **Other**,
  Unknown }

F.42.1 Detailed Description

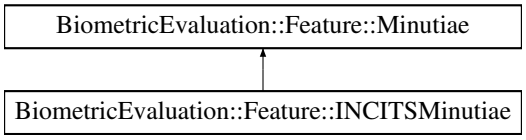
Finger and palm impression types.

F.43 BiometricEvaluation::Feature::INCITSMinutiae Class Reference

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

`#include <be_feature_incitsminutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::INCITSMinutiae:



Public Member Functions

- MinutiaeFormat::Kind getFormat () const

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

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

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

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

  Obtains the set of delta positions. The set may be empty.
- INCITSMinutiae (const MinutiaPointSet &mps, const RidgeCountItemSet &rcis, const CorePointSet &cps, const DeltaPointSet &dps)

  Construct an INCITS Minutiae object from its components.
- INCITSMinutiae ()

  Default constructor for an INCITS Minutiae object.
- void setMinutiaPoints (const MinutiaPointSet &mps)

  Mutator for the minutiae point set.
- void setRidgeCountItems (const RidgeCountItemSet &rcis)

  Mutator for the ridge count items.
- void setCorePointSet (const CorePointSet &cps)

  Mutator for the set of core points.
- void setDeltaPointSet (const DeltaPointSet &dps)

  Mutator for the set of delta points.

Static Public Attributes

- static const string **FMR_ANSI_SPEC_VERSION**
- static const string **FMR_ISO_SPEC_VERSION**
- static const string **FMR_ANSI07_SPEC_VERSION**
- static const uint8_t **FMR_SPEC_VERSION_LEN** = 4
- static const uint32_t **FED_HEADER_LENGTH** = 4
- static const uint32_t **FED_RCD_ITEM_LENGTH** = 3
- static const uint16_t **FMD_MINUTIA_TYPE_MASK** = 0xC000
- static const uint16_t **FMD_RESERVED_MASK** = 0xC000
- static const uint16_t **FMD_MINUTIA_TYPE_SHIFT** = 14
- static const uint16_t **FMD_RESERVED_SHIFT** = 14
- static const uint16_t **FMD_X_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_Y_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_MASK** = 0xC0
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_SHIFT** = 6
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_ANGLE_MASK** = 0x3F
- static const uint16_t **FMD_MIN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MAX_MINUTIA_QUALITY** = 100
- static const uint16_t **FMD_UNKNOWN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MIN_MINUTIA_ANGLE** = 0

- static const uint16_t **FMD_MAX_MINUTIA_ANGLE** = 179
- static const uint16_t **FMD_MAX_MINUTIA_ISONC_ANGLE** = 255
- static const uint16_t **FMD_MAX_MINUTIA_ISOCC_ANGLE** = 63
- static const uint16_t **FMD_ANSI_ANGLE_UNIT** = 2
- static const uint16_t **FMD_ISO_ANGLE_UNIT**
- static const uint16_t **FMD_ISOCC_ANGLE_UNIT**
- static const uint16_t **FMD_MINUTIA_TYPE_OTHER** = 0
- static const uint16_t **FMD_MINUTIA_TYPE_RIDGE_ENDING** = 1
- static const uint16_t **FMD_MINUTIA_TYPE_BIFURCATION** = 2
- static const uint16_t **FMR_MIN_FINGER_QUALITY** = 0
- static const uint16_t **FMR_MAX_FINGER_QUALITY** = 100
- static const uint16_t **ISO_UNKNOWN_FINGER_QUALITY** = 0
- static const uint16_t **FED_RESERVED** = 0x0000
- static const uint16_t **FED_RIDGE_COUNT** = 0x0001
- static const uint16_t **FED_CORE_AND_DELTA** = 0x0002
- static const uint16_t **RCE_NONSPECIFIC** = 0x00
- static const uint16_t **RCE_FOUR_NEIGHBOR** = 0x01
- static const uint16_t **RCE_EIGHT_NEIGHBOR** = 0x02
- static const uint16_t **CORE_TYPE_NONANGULAR** = 0x00
- static const uint16_t **CORE_TYPE_ANGULAR** = 0x01
- static const uint16_t **DELTA_TYPE_NONANGULAR** = 0x00
- static const uint16_t **DELTA_TYPE_ANGULAR** = 0x01

F.43.1 Detailed Description

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

The base INCTISMinutiae class is responsible for reading minutiae data points and extended data. Each minutiae point, ridge count item, core, and delta is represented in the native ANSI/INCITS format. Objects of this base class cannot be instantiated, but rather derived classes are used to represent minutiae data taken from the INCITS-derived record formats.

F.43.2 Constructor & Destructor Documentation

BiometricEvaluation::Feature::INCITSMinutiae::INCITSMinutiae (const MinutiaPointSet & *mps*, const RidgeCountItemSet & *rcis*, const CorePointSet & *cps*, const DeltaPointSet & *dps*)

Construct an INCITS Minutiae object from its components.

The buffer index must be set to the location in the buffer to start reading minutiae data points and extended data.

Parameters

in	*mps*	The set of minutiae points.
in	*rcis*	The set of ridge count items.
in	*cps*	The set of core points.
in	*dps*	The set of delta points.

F.43.3 Member Function Documentation

void BiometricEvaluation::Feature::INCITSMinutiae::setMinutiaPoints (const MinutiaPointSet & *mps*)

Mutator for the minutiae point set.

Parameters

in	*mps*	The minutiae points.

void BiometricEvaluation::Feature::INCITSMinutiae::setRidgeCountItems (const RidgeCountItemSet & *rcis*)

Mutator for the ridge count items.
Parameters

in	*rcis*	The set of ridge count items.

void BiometricEvaluation::Feature::INCITSMinutiae::setCorePointSet (const CorePointSet & *cps*)

Mutator for the set of core points.
Parameters

in	*cps*	The set of core points.

void BiometricEvaluation::Feature::INCITSMinutiae::setDeltaPointSet (const DeltaPointSet & *dps*)

Mutator for the set of delta points.
Parameters

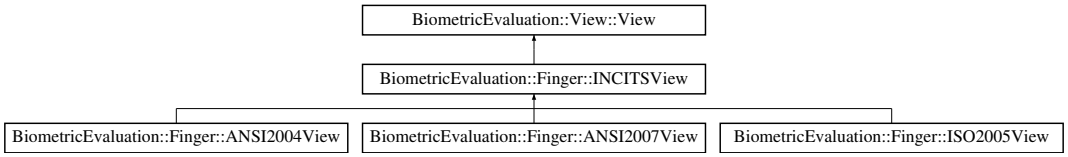
in	*dps*	The set of delta point items.

F.44 BiometricEvaluation::Finger::INCITSView Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_incitsview.h>`

Inheritance diagram for BiometricEvaluation::Finger::INCITSView:



Public Member Functions

- Feature::INCITSMinutiae getMinutiaeData () const

  Obtain the set of minutiae records.
- Finger::Position::Kind getPosition () const

  Obtain the finger position.
- Finger::Impression::Kind getImpressionType () const

  Obtain the finger impression code.
- uint32_t getQuality () const

  Obtain the finger quality value.

- uint16_t getCaptureEquipmentID () const

  Obtain the capture equipment identifier.
- bool isAppendixFCompliant () const

  Obtain the capture equipment compliance indicator for 'Appendix F'.
- tr1::shared_ptr< Image::Image > getImage () const

  Obtain the image used for the finger view.
- Image::Size getImageSize () const

  Obtain the image size.
- Image::Resolution getImageResolution () const

  Obtain the image resolution.
- uint32_t getImageDepth () const

  Obtain the image depth.
- Image::CompressionAlgorithm::Kind getCompressionAlgorithm () const

  Obtain the compression algorithm used on the image.
- Image::Resolution getScanResolution () const

  Obtain the image scan resolution.

Static Public Member Functions

- static Finger::Position::Kind throw (Error::DataError)

  Convert a finger postion code from an INCITS finger record to the common code.
- static Finger::Impression::Kind 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.

F.44.1 Detailed Description

A class to represent single finger view and derived information.

A base Finger::INCITSView object represents an INCITS/ANSI or ISO finger view. This class defines the common interface for all ANSI/ISO views as well as common implementations. Subclasses specialize this class in order to represent other versions of the ANSI/ISO specs. Objects of this class cannot be created.

F.44.2 Constructor & Destructor Documentation

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.

Exceptions

Error::DataError	Invalid record format.
Error::FileError	Could not open or read from file.

BiometricEvaluation::Finger::INCITSView::INCITSView (const Memory::uint8Array & *fmrBuffer*, const Memory::uint8Array & *firBuffer*, const uint32_t *viewNumber*) 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	*fmrBuffer*	The buffer containing the complete finger minutiae record.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.

F.44.3 Member Function Documentation

static Finger::Position::Kind BiometricEvaluation::Finger::INCITSView::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.

static Finger::Impression::Kind BiometricEvaluation::Finger::INCITSView::throw (Error::DataError) [static]

Convert a impression type code from an INCITS finger record to the common code.

Parameters

in	*incitsIMP*	A finger impression type code as defined by the INCITS standard.

Exceptions

Error::DataError	The impression type code is invalid.

Returns

The finger impression type code in common notation.

Finger::Position::Kind BiometricEvaluation::Finger::INCITSView::getPosition ( ) const

Obtain the finger position.

Returns

The finger position.

Finger::Impression::Kind BiometricEvaluation::Finger::INCITSView::getImpressionType ( ) const

Obtain the finger impression code.

Returns

The finger impression code.

uint32_t BiometricEvaluation::Finger::INCITSView::getQuality ( ) const

Obtain the finger quality value.

Returns

The finger quality value.

uint16_t BiometricEvaluation::Finger::INCITSView::getCaptureEquipmentID ( ) const

Obtain the capture equipment identifier.

Returns

The equipment ID.

bool BiometricEvaluation::Finger::INCITSView::isAppendixFCompliant ( ) const

Obtain the capture equipment compliance indicator for 'Appendix F'.

Returns

True if 'Appendix F' compliant, false otherwise.

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.

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.

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.

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.

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.

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.

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.

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.

void BiometricEvaluation::Finger::INCITSView::setMinutiaeData (const Feature::INCITSMinutiae & *fmd*) [protected]

Mutator for the Feature::INCITSMinutiae item.

Parameters

in	*fmd*	The minutiae data object.

void BiometricEvaluation::Finger::INCITSView::setPosition (const Finger::Position::Kind & *position*) [protected]

Mutator for the position.

Parameters

in	*position*	The finger position.

void BiometricEvaluation::Finger::INCITSView::setImpressionType (const Finger::Impression::Kind & *impression*) [protected]

Mutator for the impression type.

Parameters

in	*impression*	The finger impression type code.

void BiometricEvaluation::Finger::INCITSView::setQuality (uint32_t *quality*) [protected]

Mutator for the finger quality value.
Parameters

in	*quality*	The quality value.

void BiometricEvaluation::Finger::INCITSView::setViewNumber (uint32_t *viewNumber*) [protected]

Mutator for the finger view number.
Parameters

in	*viewNumber*	The view number value.

void BiometricEvaluation::Finger::INCITSView::setCaptureEquipmentID (uint16_t *id*) [protected]

Mutator for the equipment ID.
Parameters

in	*id*	The equipment ID value.

void BiometricEvaluation::Finger::INCITSView::setCBEFFProductIDs (uint16_t *owner*, uint16_t *type*) [protected]

Mutator for the CBEFF Product ID owner and type.
Parameters

in	*owner*	The CBEFF ID of the product owner.
in	*type*	The CBEFF ID of the product type.

void BiometricEvaluation::Finger::INCITSView::setAppendixFCompliance (bool *flag*) [protected]

Mutator for the Appendix F compliance indicator.
Parameters

in	*flag*	True if the capture equipment is 'Appendix F' compliant, false if not.

void BiometricEvaluation::Finger::INCITSView::setImageSize (const Image::Size & *imageSize*) [protected]

Mutator for the image size.

Parameters

in	*imageSize*	The image size object.

void BiometricEvaluation::Finger::INCITSView::setImageResolution (const Image::Resolution & *imageResolution*) [protected]

Mutator for the image resolution.
Parameters

in	*image-Resolution*	The image resolution object.

void BiometricEvaluation::Finger::INCITSView::setScanResolution (const Image::Resolution & *scanResolution*) [protected]

Mutator for the image scan resolution.
Parameters

in	*scanResolution*	The image scan resolution object.

void BiometricEvaluation::Finger::INCITSView::setImageData (const Memory::uint8Array & *imageData*) [protected]

Mutator for the image data.
Parameters

in	*imageData*	The image data object.

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.

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.

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.
in	*count*	Number of minutiae data points to read.

Exceptions

DataError	The INCITS record has invalid or missing data.

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.

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.

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::ANSI2007View, BiometricEvaluation::Finger::ISO2005View, and BiometricEvaluation::Finger::ANSI2004View.

F.45 BiometricEvaluation::Memory::IndexedBuffer Class Reference

Manage a memory buffer with an index.

`#include <be_memory_indexedbuffer.h>`

Public Member Functions

- **operator uint8_t** * ()
- uint8_t * **operator->** ()
- IndexedBuffer & **operator=** (const IndexedBuffer &other)
- IndexedBuffer ()

  Create an indexed buffer of 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.

F.45.1 Detailed Description

Manage a memory buffer with an index.

The memory buffer is treated as an array of unsigned eight bit values. This class provides safe access to the array with methods to retrieve 8/16/32/64-bit elements, or and arbitrary segment starting at the index, from the array while advancing the current index. An exception is thrown by these methods whenever the retrieval would reach beyond the size of the buffer.

The buffer can also be accessed directly by subscripting.

F.45.2 Constructor & Destructor Documentation

BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (uint8_t * *data*, uint32_t *size*)

Create an indexed buffer around an existing buffer of a given length.

An object constructed in this manner will not free the underlying data buffer.

F.45.3 Member Function Documentation

uint32_t BiometricEvaluation::Memory::IndexedBuffer::getSize ( )

Obtain the current size of the buffer.

Returns

The current buffer size.

uint32_t BiometricEvaluation::Memory::IndexedBuffer::getIndex ( )

Obtain the current index into the buffer.

Returns

The current buffer index.

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.

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.

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.

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.

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.

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.

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.

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. Can be NULL. The current index is incremented.
in	*len*	The number of elements to copy.

Exceptions

Error::DataError	The buffer is exhausted.

Returns

The number of elements copied.

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.

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

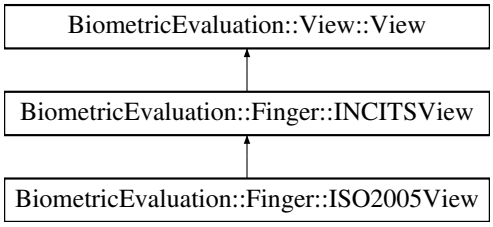
Reference to const element 'i' of the buffer.

F.46 BiometricEvaluation::Finger::ISO2005View Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_iso2005view.h>`

Inheritance diagram for BiometricEvaluation::Finger::ISO2005View:



Public Member Functions

- ISO2005View (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber) throw (Error::DataError, Error::FileError)

  Construct an ISO-2005 finger view from records contained in files.
- ISO2005View (Memory::uint8Array &fmrBuffer, Memory::uint8Array &firBuffer, const uint32_t 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.

Additional Inherited Members

F.46.1 Detailed Description

A class to represent single finger view and derived information.

A Finger::ISO2005View object represents a finger view from a ISO/IEC-2005 Finger Minutiae Record.

F.46.2 Constructor & Destructor Documentation

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.

BiometricEvaluation::Finger::ISO2005View::ISO2005View (Memory::uint8Array & *fmrBuffer,* Memory::uint8Array & *firBuffer,* const uint32_t *viewNumber*) throw Error::DataError)

Construct an ISO-2005 finger view from records contained in buffers.

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

Parameters

in	*fmrBuffer*	The buffer containing the complete finger minutiae record.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError	Invalid record format.

F.46.3 Member Function Documentation

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.

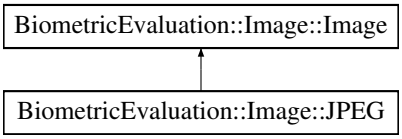
Implements BiometricEvaluation::Finger::INCITSView.

F.47 BiometricEvaluation::Image::JPEG Class Reference

A JPEG-encoded image.

`#include <be_image_jpeg.h>`

Inheritance diagram for BiometricEvaluation::Image::JPEG:



Public Member Functions

- **JPEG** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- Memory::AutoArray< uint8_t > getRawGrayscaleData (uint8_t depth=8) const throw (Error::DataError, Error::ParameterError)

  Accessor for decompressed data in grayscale.
- Memory::AutoArray< uint8_t > const throw (Error::DataError)

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

Static Public Member Functions

- static bool isJPEG (const uint8_t *data, const size_t size)
- static int **getc_skip_marker_segment** (const unsigned short marker, unsigned char **cbufptr, unsigned char *ebufptr)

Additional Inherited Members

F.47.1 Detailed Description

A JPEG-encoded image.

F.47.2 Member Function Documentation

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.

Memory::AutoArray<uint8_t> const BiometricEvaluation::Image::JPEG::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.

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

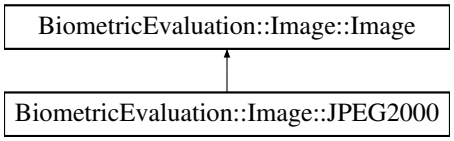
true if data appears to be a Lossy JPEG image, false otherwise

F.48 BiometricEvaluation::Image::JPEG2000 Class Reference

A JPEG-2000-encoded image.

`#include <be_image_jpeg2000.h>`

Inheritance diagram for BiometricEvaluation::Image::JPEG2000:



Public Member Functions

- JPEG2000 (const uint8_t *data, const uint64_t size, const int8_t codec=2) throw (Error::DataError, Error::StrategyError)

  Create a new JPEG2000 object.
- Memory::AutoArray< uint8_t > 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)

Additional Inherited Members

F.48.1 Detailed Description

A JPEG-2000-encoded image.

F.48.2 Constructor & Destructor Documentation

BiometricEvaluation::Image::JPEG2000::JPEG2000 (const uint8_t * *data*, const uint64_t *size*, const int8_t *codec = 2*) throw Error::DataError, Error::StrategyError)

Create a new JPEG2000 object.

Parameters

in	*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.

F.48.3 Member Function Documentation

Memory::AutoArray<uint8_t> const BiometricEvaluation::Image::JPEG2000::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.

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.

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

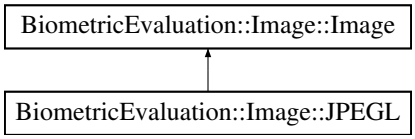
true if data appears to be a JPEG-2000 image, false otherwise.

F.49 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 > 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)

Additional Inherited Members

F.49.1 Detailed Description

A Lossless JPEG-encoded image.

F.49.2 Member Function Documentation

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.

**Memory::AutoArray<uint8_t> const BiometricEvaluation::Image::JPEGL::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.

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

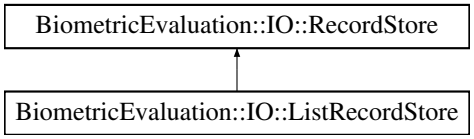
true if data appears to be a Lossless JPEG image, false otherwise.

F.50 BiometricEvaluation::IO::ListRecordStore Class Reference

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

`#include <be_io_listrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ListRecordStore:



Public Member Functions

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

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- uint64_t const **throw** (Error::StrategyError)

Static Public Attributes

- static const string SOURCERECORDSTOREPROPERTY
- static const string KEYLISTFILENAME

Additional Inherited Members

F.50.1 Detailed Description

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

ListRecordStores must be hand-crafted by first setting the 'Source Record Store', 'Type', and 'Count' properties in the .rscontrol.prop file. 'Source Record Store' is the complete path of the RecordStore containing the actual data records. Type must be 'List'. Count should match the number of entries in the file created next. Other properties are as in a "normal" RecordStore; see example below.

Second, create a file called 'KeyList.txt' in the RecordStore directory containing a list of keys, one per line.

ListRecordStores can also be created and modified with versions of rstool(1) from 2013 or later.

Example .rscontrol.prop file: Count = 10 Description = Search records for SDK TESTSDK Name = Test-LRS Type = List Source Record Store = /Users/wsalamon/sandbox/SD29.rs

Note

List RecordStores must be opened read-only.

F.50.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::ListRecordStore::ListRecordStore (const string & *name,* const string & *parentDir*) throw Error::ObjectDoesNotExist, Error::StrategyError)

Constructor, always opening read-only

BiometricEvaluation::IO::ListRecordStore::~ListRecordStore ( )

Destructor

F.50.3 Member Function Documentation

void BiometricEvaluation::IO::ListRecordStore::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 inserted.
in	*data*	The data for the record.
in	*size*	The size, in bytes, of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ListRecordStore::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.

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

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

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ListRecordStore::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.

uint64_t BiometricEvaluation::IO::ListRecordStore::length (const string & *key*) const 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.

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

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

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

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

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::ListRecordStore::setCursorAtKey (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.

void BiometricEvaluation::IO::ListRecordStore::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.

F.50.4 Member Data Documentation

const string BiometricEvaluation::IO::ListRecordStore::SOURCERECORDSTOREPROPERTY `[static]`

Property key for the source RecordStore

const string BiometricEvaluation::IO::ListRecordStore::KEYLISTFILENAME `[static]`

File name containing the list of keys

F.51 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)

F.51.1 Detailed Description

A class to represent a collection of log sheets.

F.51.2 Constructor & Destructor Documentation

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.

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.

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	*parentDir*	Where, in the file system, the cabinet is to be stored. This directory must exist.

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.

F.51.3 Member Function Documentation

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.

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.

string BiometricEvaluation::IO::LogCabinet::getName ( )

Obtain the name of the LogCabinet.
@ returns The name of the LogCabinet.

string BiometricEvaluation::IO::LogCabinet::getDescription ( )

Obtain the description of the LogCabinet.
@ returns The description of the LogCabinet.

unsigned int BiometricEvaluation::IO::LogCabinet::getCount ( )

Obtain the number of items in the LogCabinet.
@ returns The number of LogSheets manages by the cabinet.

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

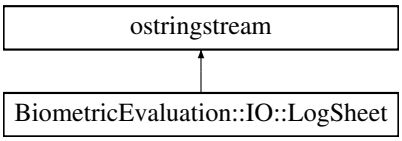
Error::ObjectDoesNotExist	The LogCabinet does not exist.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

F.52 BiometricEvaluation::IO::LogSheet Class Reference

A class to represent a single logging mechanism.

`#include <be_io_logsheet.h>`

Inheritance diagram for BiometricEvaluation::IO::LogSheet:



Public Member Functions

- LogSheet (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)

  Create a new log sheet.
- LogSheet (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Open an existing new log sheet for appending.
- virtual ~LogSheet ()
- virtual void write (const string &entry) throw (Error::StrategyError)

  Write a string as an entry to the log file.
- virtual void writeComment (const string &comment) throw (Error::StrategyError)

  Write a string as a comment to the log file.
- virtual void throw (Error::StrategyError)

  Start a new entry, causing the existing entry to be closed.
- virtual string getCurrentEntry ()

  Obtain the contents of the current entry currently under construction.
- virtual void resetCurrentEntry ()
- virtual uint32_t getCurrentEntryNumber ()

  Obtain the current entry number.
- virtual void throw (Error::StrategyError)

  Synchronize any buffered data to the underlying log file.
- void setAutoSync (bool state)
- string sequence (bool comments=false, bool trim=true, int32_t cursor=BE_LOGSHEET_SEQ_NEXT) throw (Error::FileError, Error::ObjectDoesNotExist, Error::StrategyError)

  Sequence through a LogSheet, returning one entry per invocation.

Static Public Member Functions

- static string trim (const string &entry)

  Trim delimiters from LogSheet entries.
- static void mergeLogSheets (vector< tr1::shared_ptr< LogSheet > > &logSheets) throw (Error::FileError, Error::StrategyError)

  Merge multiple LogSheets into a single LogSheet.

Static Public Attributes

- static const char CommentDelimiter = '#'
- static const char EntryDelimiter = 'E'
- static const string DescriptionTag
- static const int32_t BE_LOGSHEET_SEQ_START = 1
- static const int32_t BE_LOGSHEET_SEQ_NEXT = 2

Protected Member Functions

- LogSheet (const LogSheet &)
- LogSheet & operator= (const LogSheet &)
- void throw (Error::FileError)

  Update the cursor position of the sequence file.

Protected Attributes

- uint32_t _entryNumber
- auto_ptr< std::fstream > _theLogFile
- bool _autoSync
- tr1::shared_ptr< std::fstream > _sequenceFile
- streamoff _cursor

F.52.1 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.

F.52.2 Constructor & Destructor Documentation

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.

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.

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.

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.

virtual BiometricEvaluation::IO::LogSheet::~LogSheet ( ) `[virtual]`

Destructor

BiometricEvaluation::IO::LogSheet::LogSheet (const LogSheet &) `[protected]`

Prevent copying of LogSheet objects

F.52.3 Member Function Documentation

virtual void BiometricEvaluation::IO::LogSheet::write (const string & *entry*) throw Error::StrategyError) `[virtual]`

Write a string as an entry to the log file.

This does not affect the current log entry buffer, but does increment the entry number.

Parameters

in	*entry*	The text of the log entry.

Exceptions

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

virtual void BiometricEvaluation::IO::LogSheet::writeComment (const string & *comment*) throw Error::StrategyError) [virtual]

Write a string as a comment to the log file.

This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

in	*comment*	The text of the comment.

Exceptions

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

virtual void BiometricEvaluation::IO::LogSheet::throw (Error::StrategyError) [virtual]

Start a new entry, causing the existing entry to be closed.

Applications do not have to call this method for the first entry, however, as the stream is ready for writing upon construction.

Exceptions

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

virtual string BiometricEvaluation::IO::LogSheet::getCurrentEntry ( ) [virtual]

Obtain the contents of the current entry currently under construction.

Returns

The text of the current entry.

virtual void BiometricEvaluation::IO::LogSheet::resetCurrentEntry ( ) [virtual]

Reset the current entry buffer to the beginning.

virtual uint32_t BiometricEvaluation::IO::LogSheet::getCurrentEntryNumber ( ) [virtual]

Obtain the current entry number.

Returns

The current entry number.

virtual void BiometricEvaluation::IO::LogSheet::throw (Error::StrategyError) [virtual]

Synchronize any buffered data to the underlying log file.

This syncing is dependent on the behavior of the underlying filesystem and operating system.

Exceptions

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

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.

string BiometricEvaluation::IO::LogSheet::sequence (bool *comments* = `false`, bool *trim* = `true`, int32_t *cursor* = BE_LOGSHEET_SEQ_NEXT) throw Error::FileError, Error::ObjectDoesNotExist, Error::StrategyError)

Sequence through a LogSheet, returning one entry per invocation.

Parameters

comments	Include comments when sequencing
trim	Whether or not to include entry delimiters.
cursor	The location within the sequence to return.

Returns

The contents of the sequenced entry, as was originally given to write().

Exceptions

Error::FileError,Error	occured while performing file IO.
Error::ObjectDoesNot-Exist	The LogSheet cannot be found on disk.
Error::StrategyError	Invalid cursor position or the contents of the LogSheet is malformed.

static string BiometricEvaluation::IO::LogSheet::trim (const string & *entry*) `[static]`

Trim delimiters from LogSheet entries.

Works for comments and numbered entries.

Parameters

`in`	*entry*	The entry to trim.

Returns

Delimiter-less entry.

static void BiometricEvaluation::IO::LogSheet::mergeLogSheets (vector< tr1::shared_ptr< LogSheet > > & *logSheets*) throw Error::FileError, Error::StrategyError) `[static]`

Merge multiple LogSheets into a single LogSheet.

LogSheets 2 - n will be appended to LogSheet 1.

Parameters

logSheets	LogSheets to merge.

Exceptions

Error::FileError	Error during log sequence.
Error::StrategyError	Error during log sequence.

LogSheet& BiometricEvaluation::IO::LogSheet::operator= (const LogSheet &) [protected]

Prevent copying of LogSheet objects

void BiometricEvaluation::IO::LogSheet::throw (Error::FileError) [protected]

Update the cursor position of the sequence file.

Exceptions

Error::FileError	Error getting file position from sequence file.

F.52.4 Member Data Documentation

const char BiometricEvaluation::IO::LogSheet::CommentDelimiter = '#' [static]

Delimiter for a comment line in the log sheet.

const char BiometricEvaluation::IO::LogSheet::EntryDelimiter = 'E' [static]

Delimiter for an entry line in the log sheet.

const string BiometricEvaluation::IO::LogSheet::DescriptionTag [static]

The tag for the description string.

const int32_t BiometricEvaluation::IO::LogSheet::BE_LOGSHEET_SEQ_START = 1 [static]

Sequence from beginning

const int32_t BiometricEvaluation::IO::LogSheet::BE_LOGSHEET_SEQ_NEXT = 2 [static]

Sequence from current position

uint32_t BiometricEvaluation::IO::LogSheet::_entryNumber [protected]

Number of the current entry

auto_ptr<std::fstream> BiometricEvaluation::IO::LogSheet::_theLogFile [protected]

Stream used for writing the log file

bool BiometricEvaluation::IO::LogSheet::_autoSync [protected]

Whether or not to sync() on write()

tr1::shared_ptr<std::fstream> BiometricEvaluation::IO::LogSheet::_sequenceFile [protected]

Stream used for sequencing

streamoff BiometricEvaluation::IO::LogSheet::_cursor [protected]

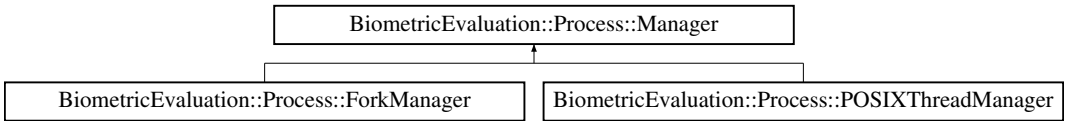
Position of the sequencer, relative to SOF

F.53 BiometricEvaluation::Process::Manager Class Reference

An interface for intranode process management classes.

`#include <be_process_manager.h>`

Inheritance diagram for BiometricEvaluation::Process::Manager:



Public Member Functions

- Manager ()

  Manager constructor.
- virtual tr1::shared_ptr < WorkerController > addWorker (tr1::shared_ptr< Worker > worker)=0

  Adds a Worker to be managed by this Manager.
- virtual uint32_t const throw (Error::StrategyError)

  Obtain the number of Workers that have exited.
- virtual uint32_t const throw (Error::StrategyError)

  Obtain the number of Workers that are still working.
- virtual void startWorkers (bool wait=true, bool communicate=false)=0 throw (Error::ObjectExists, Error::StrategyError)

  Begin Worker's work.
- virtual void startWorker (tr1::shared_ptr< WorkerController > worker, bool wait=true, bool communicate=false)=0 throw (Error::ObjectExists, Error::StrategyError)

  Start a Worker.
- virtual void throw (Error::ObjectExists)

  Reuse all Workers.
- virtual int32_t stopWorker (tr1::shared_ptr< WorkerController > worker)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Ask Worker to return as soon as possible.
- virtual bool waitForMessage (tr1::shared_ptr< WorkerController > &sender, int *nextFD=NULL, int numSeconds=-1) const

  Wait for a message from a Worker.
- virtual bool getNextMessage (tr1::shared_ptr< WorkerController > &sender, Memory::uint8Array &message, int numSeconds=-1) const throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Obtain a message from a Worker.

- virtual void broadcastMessage (Memory::uint8Array &message) const throw (Error::StrategyError)

  Send one message to all Workers.
- virtual ~Manager ()

  Manager destructor.

Public Attributes

- virtual uint32_t const

  Obtain the number of Workers this class is handling.

Protected Member Functions

- virtual void _wait ()=0

  Do not return until all spawned processes exited.

Protected Attributes

- vector< tr1::shared_ptr < WorkerController > > _workers
- vector< tr1::shared_ptr < WorkerController > > _pendingExit

F.53.1 Detailed Description

An interface for intranode process management classes.

F.53.2 Member Function Documentation

virtual tr1::shared_ptr<WorkerController> BiometricEvaluation::Process::Manager::addWorker (tr1::shared_ptr< Worker > *worker*) [pure virtual]

Adds a Worker to be managed by this Manager.

Parameters

worker	A Worker instance to run.

Returns

shared_ptr to worker.

Implemented in BiometricEvaluation::Process::ForkManager, and BiometricEvaluation::Process::POSIXThreadManager.

virtual uint32_t const BiometricEvaluation::Process::Manager::throw (Error::StrategyError) [virtual]

Obtain the number of Workers that have exited.

Returns

The number of Workers that have exited.

Exceptions

Error::StrategyError	No Workers have started working yet.

virtual uint32_t const BiometricEvaluation::Process::Manager::throw (Error::StrategyError) [virtual]

Obtain the number of Workers that are still working.

Returns

The number of Workers that are still working.

Exceptions

Error::StrategyError	No Workers have started working yet.

virtual void BiometricEvaluation::Process::Manager::startWorkers (bool *wait* = *true*, bool *communicate* = *false*) throw Error::ObjectExists, Error::StrategyError) [pure virtual]

Begin Worker's work.

Parameters

in	*wait*	Whether or not to wait for all Workers to return before returning.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	At least one Worker is already working.
Error::StrategyError	Problem starting Workers.

Implemented in BiometricEvaluation::Process::ForkManager, and BiometricEvaluation::Process::POSIXThreadManager.

virtual void BiometricEvaluation::Process::Manager::startWorker (tr1::shared_ptr< WorkerController > *worker*, bool *wait* = *true*, bool *communicate* = *false*) throw Error::ObjectExists, Error::StrategyError) [pure virtual]

Start a Worker.

Parameters

	worker	Pointer to a WorkerController that is being managed by this Manager instance.
	wait	Whether or not to wait for this Worker to exit before returning control to the caller.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

Note

Some implementations of this interface may call the system exit function from this routine. Therefore, the application's implementation of workerMain() should release all resources before returning.

Implemented in BiometricEvaluation::Process::ForkManager, and BiometricEvaluation::Process::POSIXThreadManager.

virtual void BiometricEvaluation::Process::Manager::throw (Error::ObjectExists) [virtual]

Reuse all Workers.

Exceptions

Error::ObjectExists	At least one Worker is still working.

virtual int32_t BiometricEvaluation::Process::Manager::stopWorker (tr1::shared_ptr< WorkerController > *worker*) throw Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

Ask Worker to return as soon as possible.

Parameters

worker	Pointer to the WorkerController that should be stopped.

Returns

Return code of worker.

Exceptions

Error::ObjectDoesNotExist	worker is not working.
Error::StrategyError	Problem asking worker to stop.

Implemented in BiometricEvaluation::Process::ForkManager, and BiometricEvaluation::Process::POSIXThreadManager.

virtual bool BiometricEvaluation::Process::Manager::waitForMessage (tr1::shared_ptr< WorkerController > & *sender*, int * *nextFD* = *NULL*, int *numSeconds* = *-1*) const [virtual]

Wait for a message from a Worker.

Parameters

out	*sender*	Reference to a shared pointer of the WorkerController that sent the message.
in,out	*nextFD*	Location to store a pipe that has data to read.
in	*numSeconds*	Number of seconds to wait for a message, or < 0 to block.

Returns

true if there is a Worker sending a message false otherwise or if an error occurred.

virtual bool BiometricEvaluation::Process::Manager::getNextMessage (tr1::shared_ptr< WorkerController > & *sender*, Memory::uint8Array & *message*, int *numSeconds* = *-1*) const throw Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Obtain a message from a Worker.

Parameters

out	*sender*	Reference to a shared pointer of the WorkerController that sent the message.
out	*message*	Reference to a buffer to hold the message.
in	*numSeconds*	Number of seconds to wait for a message, or < 0 to block.

Returns

true if there is a message, false otherwise.

Exceptions

Error::ObjectDoesNotExist	(Unexpected) widowed pipe.
Error::StrategyError	Error receiving message.

virtual void BiometricEvaluation::Process::Manager::broadcastMessage (Memory::uint8Array & *message*) const throw Error::StrategyError) [virtual]

Send one message to all Workers.

Parameters

message	The message to send to all Workers.

Exceptions

Error::StrategyError	Error propagated from the WorkerController.

F.53.3 Member Data Documentation

virtual uint32_t BiometricEvaluation::Process::Manager::const

Obtain the number of Workers this class is handling.

Returns

Number of Workers.

vector<tr1::shared_ptr<WorkerController> > BiometricEvaluation::Process::Manager::_workers [protected]

Workers that have been added.

vector<tr1::shared_ptr<WorkerController> > BiometricEvaluation::Process::Manager::_pendingExit [protected]

Workers that are about to exit (stop requested).

F.54 BiometricEvaluation::IO::ManifestEntry Struct Reference

`#include <be_io_archiverecstore.h>`

Public Attributes

- long offset
- uint64_t size

F.54.1 Detailed Description

Info about a single archive element

F.54.2 Member Data Documentation

long BiometricEvaluation::IO::ManifestEntry::offset

The offset from the beginning of the file/memory

uint64_t BiometricEvaluation::IO::ManifestEntry::size

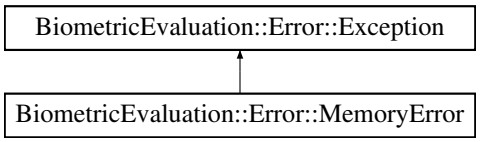
The length from offset this element spans

F.55 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)

F.55.1 Detailed Description

An error occurred when allocating an object.

F.55.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::MemoryError::MemoryError ( )

Construct a MemoryError object with the default information string.

BiometricEvaluation::Error::MemoryError::MemoryError (string *info*)

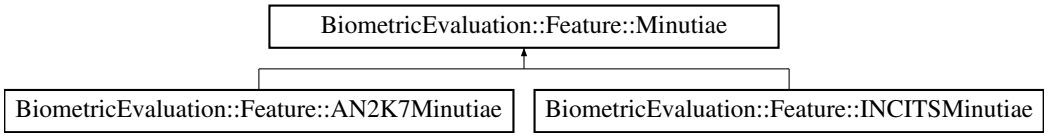
Construct a MemoryError object with an information string appended to the default information string.

F.56 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.

F.56.1 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.

F.57 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** }

F.57.1 Detailed Description

Enumerate the minutiae format standards.

F.58 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** }

F.58.1 Detailed Description

Enumerate the types of minutiae: Ridge Ending, Bifurcation, Compound, or other.

F.59 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**

F.59.1 Detailed Description

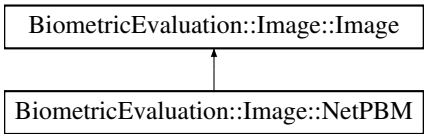
Representation of a finger minutiae data point.

F.60 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, **BinaryPortable-Bitmap** = 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 > 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.

Additional Inherited Members

F.60.1 Detailed Description

A NetPBM-encoded image.

Note

While a NetPBM file can contain more than one image, this class will only support the first image found in any file, also known as the ”plain” NetPBM format.

F.60.2 Member Function Documentation

Memory::AutoArray<uint8_t> const BiometricEvaluation::Image::NetPBM::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.

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.

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.

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

data	Buffer with line to be skipped.
offset	Position within data from which the rest of the line should be read.

Exceptions

out_of_range	End of line not encountered before end of data or on last line of data.

static void BiometricEvaluation::Image::NetPBM::skipComment (Memory::uint8Array & *data*, size_t & *offset*) throw out_of_range) [static]

Skip a block of comments in input.
Parameters

data	Buffer with comment to be skipped.
offset	Position within data from which the rest of the line should be read.

Exceptions

out_of_range	End of line not encountered before end of data or on last line of data.

static 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.

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.

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.

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

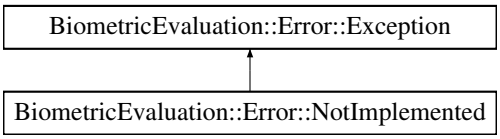
out_of_range	Error extracting a value from the bitmap.

F.61 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)

F.61.1 Detailed Description

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

F.61.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::NotImplemented::NotImplemented ( )

Construct a NotImplemented object with the default information string.

BiometricEvaluation::Error::NotImplemented::NotImplemented (string *info*)

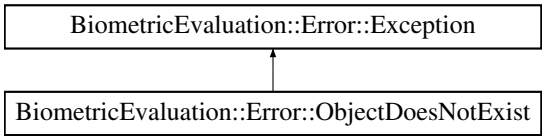
Construct a NotImplemented object with an information string appended to the default information string.

F.62 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)

F.62.1 Detailed Description

The named object does not exist.

F.62.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( )

Construct a ObjectDoesNotExist object with the default information string.

BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist (string *info*)

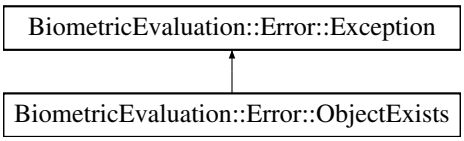
Construct a ObjectDoesNotExist object with an information string appended to the default information string.

F.63 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)

F.63.1 Detailed Description

The named object exists and will not be replaced.

F.63.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ObjectExists::ObjectExists ( )

Construct a ObjectExists object with the default information string.

BiometricEvaluation::Error::ObjectExists::ObjectExists (string *info*)

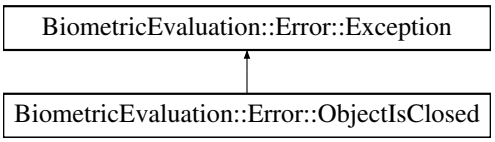
Construct a ObjectExists object with an information string appended to the default information string.

F.64 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)

F.64.1 Detailed Description

The object is closed.

F.64.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ( )

Construct a ObjectIsClosed object with the default information string.

BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed (string *info*)

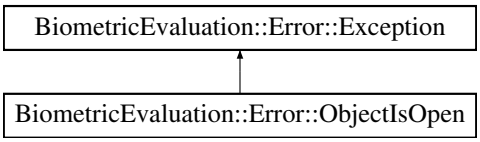
Construct a ObjectIsClosed object with an information string appended to the default information string.

F.65 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)

F.65.1 Detailed Description

The object is already opened.

F.65.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ( )

Construct a ObjectIsOpen object with the default information string.

BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen (string *info*)

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

F.66 BiometricEvaluation::Memory::OrderedMap< Key, T > Class Template Reference

`#include <be_memory_orderedmap.h>`

Public Types

- typedef std::tr1::unordered_map< Key, T > **container**
- typedef OrderedMapIterator < Key, T > **iterator**
- typedef OrderedMapConstIterator< Key, T > **const_iterator**
- typedef container::size_type **size_type**
- typedef container::value_type **value_type**
- typedef Key **key_type**
- typedef T **mapped_type**
- typedef container::key_equal **key_equal**

Public Member Functions

- OrderedMap ()
- bool push_back (const value_type &value)

  Insert an element at the end of the collection.
- void erase (iterator pos)

  Remove an element from the collection.
- void erase (const Key &key)

  Remove an element from the collection.
- iterator begin ()
- iterator end ()
- bool keyExists (const Key &key) const

  Determine if a value exists in the container.
- const OrderedMapIterator< Key, T > find (const Key &key) const

  Obtain an iterator to a particular key.
- std::tr1::shared_ptr< value_type > **find_quick** (const Key &key) const
- T & operator[ ] (const Key &key)

  Subscripting operator.
- ~OrderedMap ()

Public Attributes

- const_iterator const
- size_type const
- key_equal **const**

Friends

- class **OrderedMapIterator< Key, T >**
- class **OrderedMapConstIterator< Key, T >**

F.66.1 Detailed Description

template<class Key, class T>class BiometricEvaluation::Memory::OrderedMap< Key, T >

A map where insertion order is preserved and elements are unique.

F.66.2 Constructor & Destructor Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::OrderedMap ( )

Constructor.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::~OrderedMap ( )

Destructor

F.66.3 Member Function Documentation

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMap< Key, T >::push_back (const value_type & *value*)

Insert an element at the end of the collection.
Parameters

value	Value to insert.

Returns

Whether or not the object was inserted.

Note

Complexity: Average case: O(1), worst case O(size()).

template<class Key, class T> void BiometricEvaluation::Memory::OrderedMap< Key, T >::erase (iterator *pos*)

Remove an element from the collection.

Parameters

pos	Iterator to element at the position which should be removed.

Note

Complexity: Average case: O(1), worst case O(size()).

template<class Key, class T > void BiometricEvaluation::Memory::OrderedMap< Key, T >::erase (const Key & *key*)

Remove an element from the collection.
Parameters

pos	Key of the element to remove.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::begin ( )

Returns

Iterator at the first element of the collection.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::end ( )

Returns

Iterator beyond the last element of the collection.

template<class Key, class T > bool BiometricEvaluation::Memory::OrderedMap< Key, T >::keyExists (const Key & *key*) const

Determine if a value exists in the container.
Parameters

key	Key to search the container for.

Returns

Whether or not key exists in this container.

Complexity is O(1).

template<class Key, class T > const BiometricEvaluation::Memory::OrderedMapIterator< Key, T > BiometricEvaluation::Memory::OrderedMap< Key, T >::find (const Key & *key*) const

Obtain an iterator to a particular key.
Complexity is O(n).

template<class Key, class T > T & BiometricEvaluation::Memory::OrderedMap< Key, T >::operator[ ] (const Key & *key*)

Subscripting operator.

Parameters

key	Key used to index into the map.

Returns

Value for key, which may be a new value.

F.66.4 Member Data Documentation

template<class Key, class T> const_iterator BiometricEvaluation::Memory::OrderedMap< Key, T >::const

Returns

Iterator at the first element of the collection.
Iterator beyond the last element of the collection.

template<class Key, class T> size_type BiometricEvaluation::Memory::OrderedMap< Key, T >::const

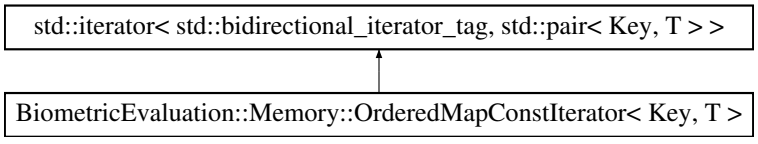
Returns

Number of elements in the collection.

F.67 BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > Class Template Reference

`#include <be_memory_orderedmap.h>`

Inheritance diagram for BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >:



Public Types

- typedef std::iterator_traits < OrderedMapConstIterator > ::reference **reference**
- typedef const std::iterator_traits < OrderedMapConstIterator > ::reference **const_reference**
- typedef std::iterator_traits < OrderedMapConstIterator > ::pointer **pointer**
- typedef const std::iterator_traits < OrderedMapConstIterator > ::pointer **const_pointer**

- typedef std::iterator_traits < OrderedMapConstIterator > ::value_type **value_type**
- typedef std::iterator_traits < OrderedMapConstIterator > ::difference_type **difference_type**

Public Member Functions

- OrderedMapConstIterator ()
- OrderedMapConstIterator (const OrderedMapIterator< Key, T > &iterator)
- ~OrderedMapConstIterator ()
- const_reference operator* () const
- const_pointer operator-> () const
- OrderedMapConstIterator & operator++ ()
- OrderedMapConstIterator & operator++ (int dummy)
- OrderedMapConstIterator & operator-- ()
- OrderedMapConstIterator & operator-- (int dummy)
- bool operator== (const OrderedMapConstIterator &rhs) const

  Test for iterator equality.
- bool operator!= (const OrderedMapConstIterator &rhs) const

  Test for iterator equality.

Friends

- class **OrderedMap< Key, T >**

F.67.1 Detailed Description

template<class Key, class T>class BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >

Const Iterator for OrderedMaps.

F.67.2 Constructor & Destructor Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::OrderedMapConstIterator ( )

Constructor

template<class Key, class T> BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::OrderedMapConstIterator (const OrderedMapIterator< Key, T > & *iterator*)

Iterator to ConstIterator converter

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::~OrderedMapConstIterator ( )

Destructor

F.67.3 Member Function Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::const_reference BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator* ( ) const

Returns

Reference to the current iterated pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::const_pointer BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator-> ( ) const

Returns

Pointer to the current iterated pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator++ ( )

Move to the next pair

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator++ (int *dummy*)

Move to the next pair

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator-- ( )

Move to the previous pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator-- (int *dummy*)

Move to the previous pair.

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator== (const OrderedMapConstIterator< Key, T > & *rhs*) const

Test for iterator equality.
Parameters

rhs	Object on the right-hand side of the expression.

Returns

Whether or not this iterator is equivalent to rhs.

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator!= (const OrderedMapConstIterator< Key, T > & *rhs*) const

Test for iterator equality.

Parameters

rhs	Object on the right-hand side of the expression.

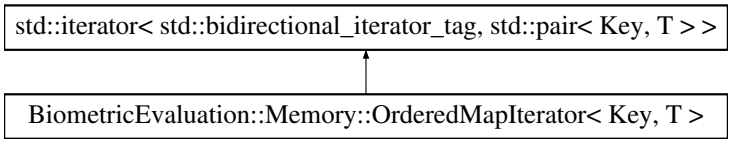
Returns

Whether or not this iterator is not equivalent to rhs.

F.68 BiometricEvaluation::Memory::OrderedMapIterator< Key, T > Class Template Reference

`#include <be_memory_orderedmap.h>`

Inheritance diagram for BiometricEvaluation::Memory::OrderedMapIterator< Key, T >:



Public Types

- typedef std::iterator_traits < OrderedMapIterator > ::reference **reference**
- typedef std::iterator_traits < OrderedMapIterator > ::pointer **pointer**
- typedef std::iterator_traits < OrderedMapIterator > ::value_type **value_type**
- typedef std::iterator_traits < OrderedMapIterator > ::difference_type **difference_type**

Public Member Functions

- OrderedMapIterator ()
- ~OrderedMapIterator ()
- reference operator* () const
- pointer operator-> () const
- OrderedMapIterator & operator++ ()
- OrderedMapIterator & operator++ (int dummy)
- OrderedMapIterator & operator-- ()
- OrderedMapIterator & operator-- (int dummy)
- bool operator== (const OrderedMapIterator &rhs) const

  Test for iterator equality.
- bool operator!= (const OrderedMapIterator &rhs) const

  Test for iterator equality.

Friends

- class **OrderedMap< Key, T >**
- class **OrderedMapConstIterator< Key, T >**

F.68.1 Detailed Description

template<class Key, class T>class BiometricEvaluation::Memory::OrderedMapIterator< Key, T >

Iterator for OrderedMaps.

F.68.2 Constructor & Destructor Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::OrderedMapIterator ( )

Constructor

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::~OrderedMapIterator ( )

Destructor

F.68.3 Member Function Documentation

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::reference BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator* ( ) const

Returns

Reference to the current iterated pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::pointer BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator-> ( ) const

Returns

Pointer to the current iterated pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator++ ( )

Move to the next pair

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator++ (int *dummy*)

Move to the next pair

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator-- ( )

Move to the previous pair.

template<class Key , class T > BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator-- (int *dummy*)

Move to the previous pair.

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator== (const OrderedMapIterator< Key, T > & *rhs*) const

Test for iterator equality.
Parameters

rhs	Object on the right-hand side of the expression.

Returns

Whether or not this iterator is equivalent to rhs.

template<class Key , class T > bool BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator!= (const OrderedMapIterator< Key, T > & *rhs*) const

Test for iterator equality.
Parameters

rhs	Object on the right-hand side of the expression.

Returns

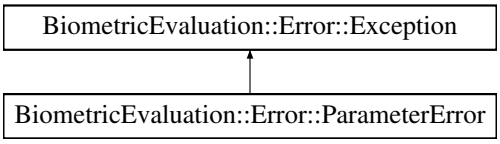
Whether or not this iterator is not equivalent to rhs.

F.69 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)

F.69.1 Detailed Description

An invalid parameter was passed to a constructor or method.

F.69.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::ParameterError::ParameterError ( )

Construct a ParameterError object with the default information string.

BiometricEvaluation::Error::ParameterError::ParameterError (string *info*)

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

F.70 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification Class Reference

Pattern classification codes.

`#include <be_feature_an2k7minutiae.h>`

Classes

- struct Entry

Public Types

- typedef struct Entry **Entry**

F.70.1 Detailed Description

Pattern classification codes.

F.71 BiometricEvaluation::Finger::PatternClassification Class Reference

Pattern classification codes.

`#include <be_finger.h>`

Public Types

- enum **Kind** {
  PlainArch = 0, **TentedArch**, **RadialLoop**, **UlnarLoop**,
  PlainWhorl, **CentralPocketLoop**, **DoubleLoop**, **AccidentalWhorl**,
  Whorl, **RightSlantLoop**, **LeftSlantLoop**, **Scar**,
  Amputation, **Unknown** }

F.71.1 Detailed Description

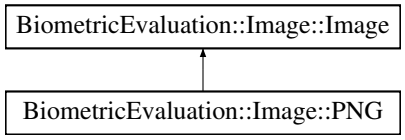
Pattern classification codes.

F.72 BiometricEvaluation::Image::PNG Class Reference

A PNG-encoded image.

`#include <be_image_png.h>`

Inheritance diagram for BiometricEvaluation::Image::PNG:



Public Member Functions

- **PNG** (const uint8_t *data, const uint64_t size) throw (Error::DataError, Error::StrategyError)
- Memory::AutoArray< uint8_t > 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)

Additional Inherited Members

F.72.1 Detailed Description

A PNG-encoded image.

F.72.2 Member Function Documentation

Memory::AutoArray<uint8_t> const BiometricEvaluation::Image::PNG::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.

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.

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

F.73 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 }

F.73.1 Detailed Description

Finger position codes.

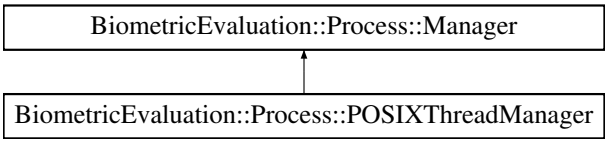
These codes match those in ANSI/NIST. Other minutiae formats may have to map codes into this set.

F.74 BiometricEvaluation::Process::POSIXThreadManager Class Reference

Manager implementation that starts Workers in POSIX threads.

`#include <be_process_posixthreadmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::POSIXThreadManager:



Public Member Functions

- POSIXThreadManager ()
- tr1::shared_ptr< WorkerController > addWorker (tr1::shared_ptr< Worker > worker)

  Adds a Worker to be managed by this Manager.
- void startWorkers (bool wait=true, bool communicate=false) throw (Error::ObjectExists, Error::StrategyError)

  Begin Worker's work.
- void startWorker (tr1::shared_ptr< WorkerController > worker, bool wait=true, bool communicate=false) throw (Error::ObjectExists, Error::StrategyError)

  Start a Worker.
- int32_t stopWorker (tr1::shared_ptr< WorkerController > workerController) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Ask Worker to exit.
- ~POSIXThreadManager ()

  ~POSIXThreadManager destructor.

Additional Inherited Members

F.74.1 Detailed Description

Manager implementation that starts Workers in POSIX threads.

F.74.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::POSIXThreadManager::POSIXThreadManager ( )

POSIXThreadManager constructor.

F.74.3 Member Function Documentation

tr1::shared_ptr<WorkerController> BiometricEvaluation::Process::POSIXThreadManager::addWorker (tr1::shared_ptr< Worker > *worker*) `[virtual]`

Adds a Worker to be managed by this Manager.

Parameters

worker	A Worker instance to run.

Returns

shared_ptr to worker.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::POSIXThreadManager::startWorkers (bool *wait* = `true`, bool *communicate* = `false`) throw Error::ObjectExists, Error::StrategyError) `[virtual]`

Begin Worker's work.
Parameters

in	*wait*	Whether or not to wait for all Workers to return before returning.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	At least one Worker is already working.
Error::StrategyError	Problem starting the Workers.

Implements BiometricEvaluation::Process::Manager.

void BiometricEvaluation::Process::POSIXThreadManager::startWorker (tr1::shared_ptr< WorkerController > *worker*, bool *wait* = `true`, bool *communicate* = `false`) throw Error::ObjectExists, Error::StrategyError) `[virtual]`

Start a Worker.
Parameters

worker	Pointer to a WorkerController that is being managed by this Manager instance.
wait	Whether or not to wait for this Worker to exit before returning control to the caller.
communicate	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists	worker is already working.
Error::StrategyError	worker is not managed by this Manager instance.

Implements BiometricEvaluation::Process::Manager.

int32_t BiometricEvaluation::Process::POSIXThreadManager::stopWorker (tr1::shared_ptr< WorkerController > *workerController*) throw Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Ask Worker to exit.
Parameters

worker-Controller	Pointer to the WorkerController that should be stopped.

Returns

Exit status of worker.

Exceptions

Error::ObjectDoesNotExist	worker is not working.
Error::StrategyError	Problem sending the signal.

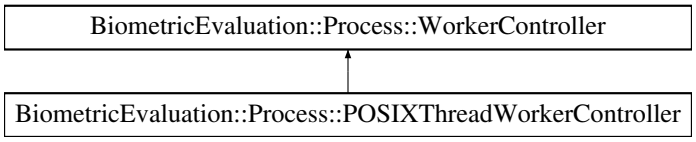
Implements BiometricEvaluation::Process::Manager.

F.75 BiometricEvaluation::Process::POSIXThreadWorkerController Class Reference

Decorated Worker returned from a Process::POSIXThreadManager.

`#include <be_process_posixthreadmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::POSIXThreadWorkerController:



Public Member Functions

- void throw (Error::ObjectExists)

  Reuse the Worker.
- ~POSIXThreadWorkerController ()

  POSIXThreadWorkerController destructor.

Public Attributes

- bool const

  Obtain whether or not Worker is working.

Friends

- class **POSIXThreadManager**

Additional Inherited Members

F.75.1 Detailed Description

Decorated Worker returned from a Process::POSIXThreadManager.

F.75.2 Member Function Documentation

void BiometricEvaluation::Process::POSIXThreadWorkerController::throw (Error::ObjectExists) `[virtual]`

Reuse the Worker.

Exceptions

Error::ObjectExists	The previously started Worker is still running.

Reimplemented from BiometricEvaluation::Process::WorkerController.

F.75.3 Member Data Documentation

bool BiometricEvaluation::Process::POSIXThreadWorkerController::const

Obtain whether or not Worker is working.

Returns

Whether or not the Worker is working.

F.76 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

F.76.1 Detailed Description

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

F.76.2 Constructor & Destructor Documentation

BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPositionCoordinate::PrintPositionCoordinate (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).

F.76.3 Member Data Documentation

FingerImageCode::Kind BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPosition-Coordinate::fingerView

Full finger view being bounded

FingerImageCode::Kind BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPosition-Coordinate::segment

Segment within full finger view bound

Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewVariableResolution::PrintPosition-Coordinate::coordinates

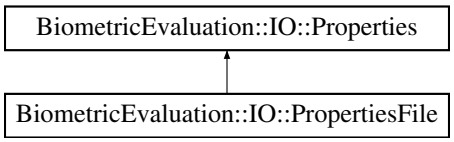
Two coordinates forming bounding box

F.77 BiometricEvaluation::IO::Properties Class Reference

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

`#include <be_io_properties.h>`

Inheritance diagram for BiometricEvaluation::IO::Properties:



Public Types

- typedef
  PropertiesMap::const_iterator const_iterator

Public Member Functions

- Properties (uint8_t mode=IO::READWRITE)

  Construct a new Properties object.
- Properties (const uint8_t *buffer, const size_t size, uint8_t mode=IO::READWRITE) throw (Error::-StrategyError)

  Construct a new Properties object from the contents of a buffer.
- virtual void setProperty (const string &property, const string &value) throw (Error::StrategyError)

  Set a property with a value.
- virtual void setPropertyFromInteger (const string &property, int64_t value) throw (Error::StrategyError)

  Set a property with an integer value.
- virtual void setPropertyFromDouble (const string &property, double value) throw (Error::StrategyError)

  Set a property with a double value.

- virtual void removeProperty (const string &property) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Remove a property.
- virtual string getProperty (const string &property) const throw (Error::ObjectDoesNotExist)

  Retrieve a property value as a string object.
- virtual int64_t getPropertyAsInteger (const string &property) const throw (Error::ObjectDoesNotExist, Error::ConversionError)

  Retrieve a property value as an integer value.
- virtual double getPropertyAsDouble (const string &property) const throw (Error::ObjectDoesNotExist)

  Retrieve a property value as a double value.
- virtual ~Properties ()

Public Attributes

- const_iterator const

  Obtain iterator to the first property.

Protected Member Functions

- void initWithBuffer (const Memory::uint8Array &buffer) throw (Error::StrategyError)

  Initialize the PropertiesMap with the contents of a properly formatted buffer.
- void initWithBuffer (const uint8_t *const buffer, size_t size) throw (Error::StrategyError)

  Initialize the PropertiesMap with the contents of a properly formatted buffer.

Protected Attributes

- uint8_t const

  Obtain the mode of the Properties object.

F.77.1 Detailed Description

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

F.77.2 Member Typedef Documentation

typedef PropertiesMap::const_iterator BiometricEvaluation::IO::Properties::const_iterator

Convenience const iterator over a Properties

F.77.3 Constructor & Destructor Documentation

BiometricEvaluation::IO::Properties::Properties (uint8_t *mode* = `IO::READWRITE`)

Construct a new Properties object.

Parameters

in	*mode*	The read/write mode of the object.

BiometricEvaluation::IO::Properties::Properties (const uint8_t * *buffer*, const size_t *size*, uint8_t *mode* = *IO::READWRITE*) throw Error::StrategyError)

Construct a new Properties object from the contents of a buffer.

The format of the buffer can be seen in PropertiesFile.

Parameters

in	*buffer*	A buffer that contains the contents of a Property file.
in	*size*	The size of buffer.
in	*mode*	The read/write mode of the object.

Exceptions

Error::StrategyError	A line in the properties file is malformed.

virtual BiometricEvaluation::IO::Properties::~Properties ( ) `[virtual]`

Destructor

F.77.4 Member Function Documentation

virtual void BiometricEvaluation::IO::Properties::setProperty (const string & *property*, const string & *value*) throw Error::StrategyError) `[virtual]`

Set a property with a value.

Both the property and value will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise, the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError	The Properties object is read-only.

virtual void BiometricEvaluation::IO::Properties::setPropertyFromInteger (const string & *property*, int64_t *value*) throw Error::StrategyError) `[virtual]`

Set a property with an integer value.

The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError	The Properties object is read-only.

virtual void BiometricEvaluation::IO::Properties::setPropertyFromDouble (const string & *property*, double *value*) throw Error::StrategyError) `[virtual]`

Set a property with a double value.

The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError	The Properties object is read-only.

virtual void BiometricEvaluation::IO::Properties::removeProperty (const string & *property*) throw Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Remove a property.
Parameters

in	*property*	The name of the property to set.

Exceptions

Error::ObjectDoesNotExist	The named property does not exist.
Error::StrategyError	The Properties object is read-only.

virtual string BiometricEvaluation::IO::Properties::getProperty (const string & *property*) const throw Error::ObjectDoesNotExist) [virtual]

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.

virtual int64_t BiometricEvaluation::IO::Properties::getPropertyAsInteger (const string & *property*) const throw Error::ObjectDoesNotExist, Error::ConversionError) [virtual]

Retrieve a property value as an integer value.

Integer value strings for properties can represent either decimal or hexadecimal values, which must be preceded with either ”0x” or ”0X”.
Parameters

in	*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.

virtual double BiometricEvaluation::IO::Properties::getPropertyAsDouble (const string & *property*) const throw Error::ObjectDoesNotExist) [virtual]

Retrieve a property value as a double value.

Parameters

in	*property*	The name of the property to get.

Exceptions

Error::ObjectDoesNotExist	The named property does not exist.

void BiometricEvaluation::IO::Properties::initWithBuffer (const Memory::uint8Array & *buffer*) throw Error::StrategyError) [protected]

Initialize the PropertiesMap with the contents of a properly formatted buffer.

This method ensures that the PropertiesMap contains only the properties found within the buffer.

Parameters

buffer	Contents of a properties file.

Exceptions

Error::StrategyError	A line of the buffer is malformed.

void BiometricEvaluation::IO::Properties::initWithBuffer (const uint8_t *const *buffer*, size_t *size*) throw Error::StrategyError) [protected]

Initialize the PropertiesMap with the contents of a properly formatted buffer.

This method ensures that the PropertiesMap contains only the properties found within the buffer.

Parameters

buffer	Contents of a properties file.
size	Size of the buffer.

Exceptions

Error::StrategyError	A line of the buffer is malformed.

F.77.5 Member Data Documentation

const_iterator BiometricEvaluation::IO::Properties::const

Obtain iterator to the first property.

Obtain iterator to one past the last property.

Returns

Iterator to first property.
Iterator one past the last property.

uint8_t BiometricEvaluation::IO::Properties::const [protected]

Obtain the mode of the Properties object.

Returns

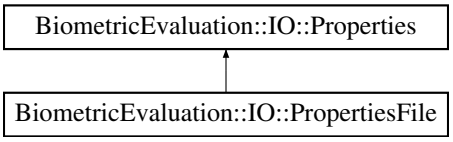
Mode (IO::READONLY or IO::READWRITE)

F.78 BiometricEvaluation::IO::PropertiesFile Class Reference

A Properties object persisted in an file on disk.

`#include <be_io_propertiesfile.h>`

Inheritance diagram for BiometricEvaluation::IO::PropertiesFile:



Public Member Functions

- PropertiesFile (const string &filename, uint8_t mode=IO::READWRITE) throw (Error::FileError, Error::StrategyError)

  Construct a new Properties object from an existing or to be created properties file. The constructor will create the file when it does not exist.
- void throw (Error::FileError, Error::StrategyError)

  Write the properties to the underlying file, synchronizing the in-memory and on-disk versions.
- void changeName (const string &filename) throw (Error::StrategyError)

  Change the name of the Properties, which means changing the name of the underlying file that stores the properties. The empty string ("") can be used to indicate no backing file.
- ~PropertiesFile ()

Additional Inherited Members

F.78.1 Detailed Description

A Properties object persisted in an file on disk.

An example file might look like this:

```
*     Name = John Smith
*     Age = 32
*     Favorite Hex Number = 0xffff
*
```

For property keys and values, leading and trailing whitespace is removed, therefore the call

```
*     props->setProperty("  My property   ", "   A Value  ");
*
```

results in an entry in the property file as

```
*     My property = A value
*
```

Therefore, the property names "Foo", " Foo", "Foo " are equivalent.

F.78.2 Constructor & Destructor Documentation

BiometricEvaluation::IO::PropertiesFile::PropertiesFile (const string & *filename,* uint8_t *mode* = *IO::READWRITE*) throw Error::FileError, Error::StrategyError)

Construct a new Properties object from an existing or to be created properties file. The constructor will create the file when it does not exist.

Parameters

in	*filename*	The name of the file to store the properties.
in	*mode*	The read/write mode of the object.

Exceptions

Error::StrategyError	A line in the properties file is malformed.
Error::FileError	An error occurred when using the underlying storage system.

BiometricEvaluation::IO::PropertiesFile::~PropertiesFile ( )

Destructor

F.78.3 Member Function Documentation

void BiometricEvaluation::IO::PropertiesFile::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.

void BiometricEvaluation::IO::PropertiesFile::changeName (const string & *filename*) throw Error::StrategyError)

Change the name of the Properties, which means changing the name of the underlying file that stores the properties. The empty string ("") can be used to indicate no backing file.

Note

No check is made that the file is writeable at this time.

Parameters

in	*filename*	The name of the properties file.

Exceptions

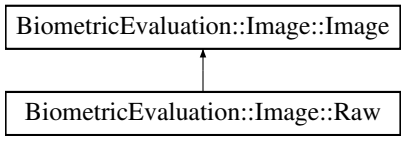
Error::StrategyError	The object is read-only.

F.79 BiometricEvaluation::Image::Raw Class Reference

An image with no encoding or compression.

`#include <be_image_raw.h>`

Inheritance diagram for BiometricEvaluation::Image::Raw:



Public Member Functions

- **Raw** (const uint8_t *data, const uint64_t size, const Size dimensions, const unsigned int depth, const Resolution resolution)
- Memory::AutoArray< uint8_t > 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.

Public Attributes

- Memory::AutoArray< uint8_t > **const**

Additional Inherited Members

F.79.1 Detailed Description

An image with no encoding or compression.

F.79.2 Member Function Documentation

Memory::AutoArray<uint8_t> const BiometricEvaluation::Image::Raw::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.

Memory::AutoArray<uint8_t> BiometricEvaluation::Image::Raw::getRawGrayscaleData (uint8_t *depth* = `8`) const throw Error::DataError, Error::ParameterError) `[virtual]`

Accessor for decompressed data in grayscale.

Parameters

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).

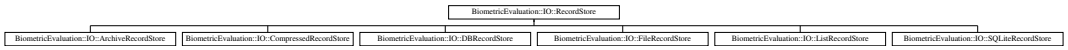
Implements BiometricEvaluation::Image::Image.

F.80 BiometricEvaluation::IO::RecordStore Class Reference

A class to represent a data storage mechanism.

`#include <be_io_recordstore.h>`

Inheritance diagram for BiometricEvaluation::IO::RecordStore:



Public Member Functions

- RecordStore (const string &name, const string &description, const string &type, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- RecordStore (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- string getName () const
- string getDescription () const
- unsigned int getCount () const
- virtual void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- virtual void changeDescription (const string &description) throw (Error::StrategyError)
- virtual uint64_t getSpaceUsed () const throw (Error::StrategyError)

  Obtain real storage utilization.
- virtual void sync () const throw (Error::StrategyError)
- virtual void insert (const string &key, const void *const data, const uint64_t size)=0 throw (Error::ObjectExists, Error::StrategyError)
- virtual void insert (const string &key, const Memory::uint8Array &data) throw (Error::ObjectExists, Error::StrategyError)
- virtual void remove (const string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t read (const string &key, void *const data) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t read (const string &key, Memory::uint8Array &data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Read a complete record from a store.
- virtual void replace (const string &key, const void *const data, const uint64_t size)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void replace (const string &key, const Memory::uint8Array &data) throw (Error::ObjectDoesNotExist, Error::StrategyError)

- virtual uint64_t length (const string &key) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void flush (const string &key) const =0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t sequence (string &key, void *const data=NULL, int cursor=BE_RECSTORE_SEQ_NEXT)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Sequence through a RecordStore, returning the key/data pairs.
- virtual uint64_t sequence (string &key, Memory::uint8Array &data, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Sequence through a RecordStore, returning the key/data pairs.
- virtual void setCursorAtKey (string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual bool containsKey (const string &key) const

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

Static Public Member Functions

- static tr1::shared_ptr < RecordStore > openRecordStore (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Open an existing RecordStore and return a managed pointer to the the object representing that store.
- static tr1::shared_ptr < RecordStore > createRecordStore (const string &name, const string &description, const string &type, const string &destDir) throw (Error::ObjectExists, Error::StrategyError)

  Create a new RecordStore and return a managed pointer to the the object representing that store.
- static void removeRecordStore (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- static void mergeRecordStores (const string &mergedName, const string &mergedDescription, const string &parentDir, const string &type, const vector< string > &path) throw (Error::ObjectExists, Error::StrategyError)

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

Static Public Attributes

- static const string INVALIDKEYCHARS
- static const char KEY_SEGMENT_SEPARATOR = '&'
- static const uint64_t KEY_SEGMENT_START = 1
- static const string CONTROLFILENAME
- static const string NAMEPROPERTY
- static const string DESCRIPTIONPROPERTY
- static const string COUNTPROPERTY
- static const string TYPEPROPERTY
- static const string BERKELEYDBTYPE
- static const string ARCHIVETYPE
- static const string FILETYPE
- static const string SQLITETYPE
- static const string COMPRESSEDTYPE
- static const string LISTTYPE
- static const string DEFAULTTYPE
- static const string RSREADONLYERROR
- static const int BE_RECSTORE_SEQ_START = 1
- static const int BE_RECSTORE_SEQ_NEXT = 2

Protected Member Functions

- uint8_t **getMode** () const
- string **getDirectory** () const
- string **getParentDirectory** () const
- string **canonicalName** (const string &name) const
- int **getCursor** () const
- void **setCursor** (int cursor)
- bool **validateKeyString** (const string &key) const
- void setProperties (const tr1::shared_ptr< IO::Properties > properties) throw (Error::StrategyError)

  Replace existing Properties in RecordStore Control File.

Static Protected Member Functions

- static string genKeySegName (const string &key, const uint64_t segnum)

  Generate key segment names.

Protected Attributes

- tr1::shared_ptr< IO::Properties > const

  Obtain a copy of the Properties object.

F.80.1 Detailed Description

A class to represent a data storage mechanism.

A RecordStore is an abstraction that associates keys with a specific record. Implementations of this abstraction can store the records in any format supported by the operating system, such as files or databases, rooted in the file system.

Certain characters are prohibited in the key string. See IO::RecordStore::INVALIDKEYCHARS. A key string cannot begin with the space character.

See Also

IO::ArchiveRecordStore, IO::DBRecordStore, IO::FileRecordStore.

F.80.2 Constructor & Destructor Documentation

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.

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.

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.

Exceptions

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

F.80.3 Member Function Documentation

string BiometricEvaluation::IO::RecordStore::getName ( ) const

Return the name of the RecordStore.

Returns

The RecordStore's name.

string BiometricEvaluation::IO::RecordStore::getDescription ( ) const

Obtain a textual description of the RecordStore.

Returns

The RecordStore's description.

unsigned int BiometricEvaluation::IO::RecordStore::getCount ( ) const

Obtain the number of items in the RecordStore.

Returns

The number of items in the RecordStore.

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::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::ListRecordStore, BiometricEvaluation::IO::FileRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

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.

Reimplemented in BiometricEvaluation::IO::SQLiteRecordStore.

virtual uint64_t BiometricEvaluation::IO::RecordStore::getSpaceUsed ( ) const throw Error::StrategyError) [virtual]

Obtain real storage utilization.

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

virtual void BiometricEvaluation::IO::RecordStore::sync ( ) const throw Error::StrategyError) [virtual]

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented in BiometricEvaluation::IO::ArchiveRecordStore, and BiometricEvaluation::IO::DBRecordStore.

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 inserted.
in	*data*	The data for the record.
in	*size*	The size, in bytes, of the record.

Exceptions

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

Implemented in BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual void BiometricEvaluation::IO::RecordStore::insert (const string & *key*, const Memory::uint8Array & *data*) throw Error::ObjectExists, Error::StrategyError) [virtual]

Insert a record into the store.
Parameters

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

Exceptions

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

virtual 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::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual uint64_t BiometricEvaluation::IO::RecordStore::read (const string & *key*, void *const *data*) const throw Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

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

in	*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::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual uint64_t BiometricEvaluation::IO::RecordStore::read (const string & *key*, Memory::uint8Array & *data*) const throw Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

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

virtual void BiometricEvaluation::IO::RecordStore::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::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual void BiometricEvaluation::IO::RecordStore::replace (const string & *key*, const Memory::uint8Array & *data*) throw Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Replace a complete record in a RecordStore.

Parameters

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

Exceptions

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

virtual uint64_t BiometricEvaluation::IO::RecordStore::length (const string & *key*) const 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::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual void BiometricEvaluation::IO::RecordStore::flush (const string & *key*) const throw Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implemented in BiometricEvaluation::IO::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::FileRecordStore, BiometricEvaluation::IO::ListRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual uint64_t BiometricEvaluation::IO::RecordStore::sequence (string & *key*, void *const *data = *NULL*, int *cursor* = BE_RECSTORE_SEQ_NEXT) throw Error::ObjectDoesNotExist, Error::StrategyError) [pure virtual]

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

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

Parameters

out	*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::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::ListRecordStore, BiometricEvaluation::IO::FileRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

virtual uint64_t BiometricEvaluation::IO::RecordStore::sequence (string & *key*, Memory::uint8Array & *data*, int *cursor* = BE_RECSTORE_SEQ_NEXT) throw Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

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

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

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

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

virtual void BiometricEvaluation::IO::RecordStore::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::CompressedRecordStore, BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, BiometricEvaluation::IO::ListRecordStore, BiometricEvaluation::IO::FileRecordStore, and BiometricEvaluation::IO::SQLiteRecordStore.

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.

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.

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.

static void BiometricEvaluation::IO::RecordStore::mergeRecordStores (const string & *mergedName*, const string & *mergedDescription*, const string & *parentDir*, const string & *type*, const vector< string > & *path*) throw Error::ObjectExists, Error::StrategyError) `[static]`

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

Parameters

in	*mergedName*	The name of the new RecordStore that will be created.
in	*merged-Description*	The text used to describe the RecordStore.
in	*parentDir*	Where the new RecordStore should be rooted.
in	*type*	The type of RecordStore that mergedName should be.
in	*path*	Vector of string paths to RecordStores to open. These point to the Record-Stores that will be merged.

Exceptions

Error::ObjectExists	A RecordStore with mergedNamed in parentDir already exists.
Error::StrategyError	An error occurred when using the underlying storage system.

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

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

key	The key to locate.

Returns

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

static string BiometricEvaluation::IO::RecordStore::genKeySegName (const string & *key*, const uint64_t *segnum*) [static], [protected]

Generate key segment names.
Parameters

key	Base key name.
segnum	Segment number for key (zero based).

Returns

Key segment name.

void BiometricEvaluation::IO::RecordStore::setProperties (const tr1::shared_ptr< IO::Properties > *properties*) throw Error::StrategyError) [protected]

Replace existing Properties in RecordStore Control File.

Existing properties will be updated. RecordStore core properties will be ignored.
Parameters

in	*properties*	Shared pointer to Properties object.

Exceptions

Error::StrategyError	RecordStore was opened READONLY.

F.80.4 Member Data Documentation

const string BiometricEvaluation::IO::RecordStore::INVALIDKEYCHARS [static]

The set of prohibited characters in a key: '/', '\', '*', '&'

const char BiometricEvaluation::IO::RecordStore::KEY_SEGMENT_SEPARATOR = '&' [static]

Character used to separate key segments

const uint64_t BiometricEvaluation::IO::RecordStore::KEY_SEGMENT_START = 1 [static]

First segment number of a segmented record

const string BiometricEvaluation::IO::RecordStore::CONTROLFILENAME [static]

The name of the control file, a properties list

const string BiometricEvaluation::IO::RecordStore::NAMEPROPERTY [static]

Property key for name of the RecordStore

const string BiometricEvaluation::IO::RecordStore::DESCRIPTIONPROPERTY [static]

Property key for description of the RecordStore

const string BiometricEvaluation::IO::RecordStore::COUNTPROPERTY [static]

Property key for the number of store items

const string BiometricEvaluation::IO::RecordStore::TYPEPROPERTY [static]

Property key for the type of RecordStore

const string BiometricEvaluation::IO::RecordStore::BERKELEYDBTYPE [static]

DBRecordStore type

const string BiometricEvaluation::IO::RecordStore::ARCHIVETYPE [static]

ArchiveRecordStore type

const string BiometricEvaluation::IO::RecordStore::FILETYPE [static]

FileRecordStore type

const string BiometricEvaluation::IO::RecordStore::SQLITETYPE [static]

SQLiteRecordStore type

const string BiometricEvaluation::IO::RecordStore::COMPRESSEDTYPE [static]

CompressedRecordStore type

const string BiometricEvaluation::IO::RecordStore::LISTTYPE [static]

ListRecordStore type

const string BiometricEvaluation::IO::RecordStore::DEFAULTTYPE [static]

Default RecordStore

const string BiometricEvaluation::IO::RecordStore::RSREADONLYERROR [static]

Message for READONLY RecordStore modification

const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_START = 1 [static]

Tell sequence() to sequence from beginning

const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_NEXT = 2 [static]

Tell sequence to sequence from current position

tr1::shared_ptr<IO::Properties> BiometricEvaluation::IO::RecordStore::const [protected]

Obtain a copy of the Properties object.
RecordStore core properties will be excluded.

Returns

Shared pointer to Properties object that may be modified.

F.81 BiometricEvaluation::View::AN2KView::RecordType Class Reference

The type of AN2K record.
`#include <be_view_an2kview.h>`

Public Types

- enum **Kind** {
  Type_1 = 1, **Type_2** = 2, **Type_3** = 3, **Type_4** = 4,
  Type_5 = 5, **Type_6** = 6, **Type_7** = 7, **Type_8** = 8,
  Type_9 = 9, **Type_10** = 10, **Type_11** = 11, **Type_12** = 12,
  Type_13 = 13, **Type_14** = 14, **Type_15** = 15, **Type_16** = 16,
  Type_17 = 17, **Type_99** = 99 }

F.81.1 Detailed Description

The type of AN2K record.

F.82 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

F.82.1 Detailed Description

A structure to represent the resolution of an image.

F.82.2 Member Enumeration Documentation

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

F.82.3 Constructor & Destructor Documentation

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

F.82.4 Member Data Documentation

double BiometricEvaluation::Image::Resolution::xRes

Resolution along the X-axis

double BiometricEvaluation::Image::Resolution::yRes

Resolution along the Y-axis

Kind BiometricEvaluation::Image::Resolution::units

Units in which xRes and yRes are represented

F.83 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 }

F.83.1 Detailed Description

Enumerate the types of extraction methods for ridge counts.

F.84 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**

F.84.1 Detailed Description

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

F.85 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

- 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

F.85.1 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.

F.85.2 Constructor & Destructor Documentation

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).

Exceptions

Error::ParameterError	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

F.85.3 Member Function Documentation

BiometricEvaluation::Error::SignalManager::throw (Error::StrategyError)

Construct a new SignalManager object with the default signal handling: SIGSEGV and SIGBUS.

Exceptions

Error::StrategyError	Could not register the signal handler.

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.).

void BiometricEvaluation::Error::SignalManager::clearSignalSet ( )

Clear all signal handling.

void BiometricEvaluation::Error::SignalManager::setDefaultSignalSet ( )

Set the default signals this object will manage: SIGSEGV and SIGBUS.

bool BiometricEvaluation::Error::SignalManager::sigHandled ( )

Indicate whether a signal was handled.

Returns

true if a signal was handled, false otherwise.

void BiometricEvaluation::Error::SignalManager::start ( ) 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.

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.

void BiometricEvaluation::Error::SignalManager::setSigHandled ( )

Set a flag to indicate a signal was handled.

void BiometricEvaluation::Error::SignalManager::clearSigHandled ( )

Clear the indication that a signal was handled.

F.85.4 Member Data Documentation

bool BiometricEvaluation::Error::SignalManager::_canSigJump `[static]`

Flag indicating can jump after handling a signal.

Note

Should not be directly used by applications.

sigjmp_buf BiometricEvaluation::Error::SignalManager::_sigJumpBuf `[static]`

The jump buffer used by the signal handler.

Note

Should not be directly used by applications.

F.86 BiometricEvaluation::Image::Size Struct Reference

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

`#include <be_image.h>`

Public Member Functions

- Size (const uint32_t xSize=0, const uint32_t ySize=0)

  Create a Size struct.

Public Attributes

- uint32_t xSize
- uint32_t ySize

F.86.1 Detailed Description

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

F.86.2 Constructor & Destructor Documentation

BiometricEvaluation::Image::Size::Size (const uint32_t *xSize* = *0*, const uint32_t *ySize* = *0*)

Create a Size struct.

Parameters

in	*xSize*	Number of pixels on the X-axis
in	*ySize*	Number of pixels on the Y-axis

F.86.3 Member Data Documentation

uint32_t BiometricEvaluation::Image::Size::xSize

Number of pixels on the X-axis

uint32_t BiometricEvaluation::Image::Size::ySize

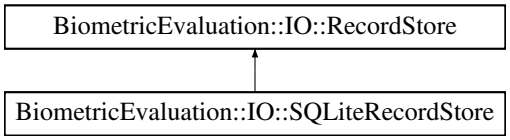
Number of pixels on the Y-axis

F.87 BiometricEvaluation::IO::SQLiteRecordStore Class Reference

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

`#include <be_io_sqliterecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::SQLiteRecordStore:



Public Member Functions

- **SQLiteRecordStore** (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- **SQLiteRecordStore** (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- void changeDescription (const string &description) throw (Error::StrategyError)
- uint64_t const **throw** (Error::StrategyError)
- void insert (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
- void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t read (const string &key, void *const data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void replace (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t length (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void flush (const string &key) const throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t sequence (string &key, void *const data=NULL, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Sequence through a RecordStore, returning the key/data pairs.
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Protected Member Functions

- void const throw (Error::StrategyError)

  Convert an SQLite error into a StrategyError.
- void throw (Error::StrategyError)

  Create the tables needed to store key->value pairs in SQLite.
- bool validateKeyValueTable (const string &table) throw (Error::StrategyError)

  Confirm that a key->value table exists with the proper schema.
- void createKeyValueTable (const string &table) throw (Error::StrategyError)

  Create a tables needed to store key->value pairs in SQLite.
- bool throw (Error::StrategyError)

  Confirm that the schema of the opened SQLite database is compatible.
- uint64_t readSegments (const string &key, void *const data) const throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Select a row from the RecordStore.
- void throw (Error::StrategyError)

  Perform SQLite cleanup routines.

Additional Inherited Members

F.87.1 Detailed Description

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

F.87.2 Member Function Documentation

void BiometricEvaluation::IO::SQLiteRecordStore::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.

void BiometricEvaluation::IO::SQLiteRecordStore::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.

Reimplemented from BiometricEvaluation::IO::RecordStore.

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

Insert a record into the store.
Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::SQLiteRecordStore::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.

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

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

in	*key*	The key of the record to be read.
in	*data*	Pointer to where the data is to be written.

Returns

The size of the record.

Exceptions

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

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

Implements BiometricEvaluation::IO::RecordStore.

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

Replace a complete record in a store.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

uint64_t BiometricEvaluation::IO::SQLiteRecordStore::length (const string & *key*) const 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.

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

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

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

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

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

Parameters

out	*key*	The key of the currently sequenced record.
in	*data*	Pointer to where the data is to be written. Applications can set data to NULL to indicate only the key is wanted.
in	*cursor*	The location within the sequence of the key/data pair to return.

Returns

The length of the record currently in sequence.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

void BiometricEvaluation::IO::SQLiteRecordStore::setCursorAtKey (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.

void const BiometricEvaluation::IO::SQLiteRecordStore::throw (Error::StrategyError) [protected]

Convert an SQLite error into a StrategyError.

Exceptions

Error::StrategyError	Always thrown with the textual description of the last error condition.

void BiometricEvaluation::IO::SQLiteRecordStore::throw (Error::StrategyError) [protected]

Create the tables needed to store key->value pairs in SQLite.

Exceptions

Error::StrategyError	Error executing SQL commands.

bool BiometricEvaluation::IO::SQLiteRecordStore::validateKeyValueTable (const string & *table*) throw Error::StrategyError) [protected]

Confirm that a key->value table exists with the proper schema.

Parameters

table	Name of the table to check.

Returns

Whether or not the table exists with the proper schema.

Exceptions

Error::StrategyError	Error compiling SQL.

void BiometricEvaluation::IO::SQLiteRecordStore::createKeyValueTable (const string & *table*) throw Error::StrategyError) [protected]

Create a tables needed to store key->value pairs in SQLite.

Parameters

table	Name of the table to create.

Exceptions

Error::StrategyError	Error executing SQL commands.

bool BiometricEvaluation::IO::SQLiteRecordStore::throw (Error::StrategyError) [protected]

Confirm that the schema of the opened SQLite database is compatible.

Returns

Whether or not the schema of the opened SQLite database is compatible with this object.

Exceptions

Error::StrategyError	Error compiling SQL.

uint64_t BiometricEvaluation::IO::SQLiteRecordStore::readSegments (const string & *key*, void *const *const *data*) const throw Error::ObjectDoesNotExist, Error::StrategyError) [protected]

Select a row from the RecordStore.

Parameters

key	Key of the row to select.
data	If not NULL, deep copy the record for key into data.

Exceptions

Error::ObjectDoesNotExist	Key does not exist in RecordStore.
Error::StrategyError	Error executing SQL commands.

Returns

Size of key's record.

void BiometricEvaluation::IO::SQLiteRecordStore::throw (Error::StrategyError) [protected]

Perform SQLite cleanup routines.

- Finalize the sequencer statement
- Close the SQLite database handle

Exceptions

Error::StrategyError	Bad return code from SQLite during cleanup.

F.88 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 ()
- 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 ()

F.88.1 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.

F.88.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::Statistics::Statistics ( )

Constructor with no parameters.

F.88.3 Member Function Documentation

BiometricEvaluation::Process::Statistics::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.

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.

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.

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.

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.

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.

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.

void BiometricEvaluation::Process::Statistics::callStatistics_logStats ( )

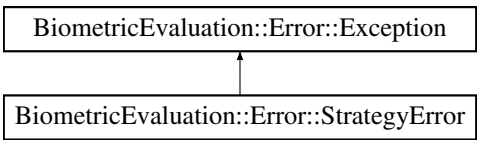
Helper function in C++ space that has access to this object, and is called from C space by the logging thread. Applications should not call this function.

F.89 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)

F.89.1 Detailed Description

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

F.89.2 Constructor & Destructor Documentation

BiometricEvaluation::Error::StrategyError::StrategyError ( )

Construct a StrategyError object with the default information string.

BiometricEvaluation::Error::StrategyError::StrategyError (string *info*)

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

F.90 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)

F.90.1 Detailed Description

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

Applcations wrap the block of code in the Timer::start() and Timer::stop() calls, then use Timer::elapsed() to obtain the calculated time of the operation.

Warning

Timers are not threadsafe and should only be used to time operations within the same thread.

F.90.2 Constructor & Destructor Documentation

BiometricEvaluation::Time::Timer::Timer ( )

Constructor for the Timer object.

F.90.3 Member Function Documentation

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.

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.

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

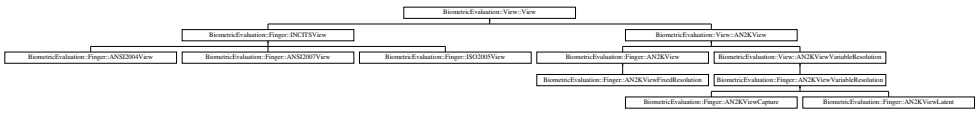
Error::StrategyError	This object is currently timing an operation or an error occurred when obtaining timing information.

F.91 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.

F.91.1 Detailed Description

A class to represent single biometric element view.

Included in a view is the biometric image and any derived information, such as minutiae points.

F.91.2 Member Function Documentation

virtual tr1::shared_ptr<Image::Image> BiometricEvaluation::View::View::getImage ( ) const [pure virtual]

Obtain the image used for the finger view.

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

Implemented in BiometricEvaluation::View::AN2KView, and BiometricEvaluation::Finger::INCITSView.

virtual Image::Size BiometricEvaluation::View::View::getImageSize ( ) const [pure virtual]

Obtain the image size.

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

Implemented in BiometricEvaluation::View::AN2KView, and BiometricEvaluation::Finger::INCITSView.

virtual Image::Resolution BiometricEvaluation::View::View::getImageResolution ( ) const [pure virtual]

Obtain the image resolution.

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

Implemented in BiometricEvaluation::View::AN2KView, and BiometricEvaluation::Finger::INCITSView.

virtual uint32_t BiometricEvaluation::View::View::getImageDepth ( ) const [pure virtual]

Obtain the image depth.

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

Implemented in BiometricEvaluation::View::AN2KView, and BiometricEvaluation::Finger::INCITSView.

virtual Image::CompressionAlgorithm::Kind BiometricEvaluation::View::View::getCompressionAlgorithm ( ) const [pure virtual]

Obtain the compression algorithm used on the image.

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

Implemented in BiometricEvaluation::View::AN2KView, and BiometricEvaluation::Finger::INCITSView.

virtual Image::Resolution BiometricEvaluation::View::View::getScanResolution ( ) const [pure virtual]

Obtain the image scan resolution.

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

Implemented in BiometricEvaluation::View::AN2KView, and BiometricEvaluation::Finger::INCITSView.

F.92 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

- throw (Error::NotImplemented, Error::ParameterError)
- void setInterval (uint64_t interval)
- void start () throw (Error::StrategyError)
- void stop () throw (Error::StrategyError)
- bool expired ()
- void setCanSigJump ()
- void clearCanSigJump ()
- void setExpired ()
- void clearExpired ()

Static Public Attributes

- static const uint8_t PROCESSTIME = 0
- static const uint8_t REALTIME = 1
- static bool **_canSigJump**
- static sigjmp_buf **_sigJumpBuf**

F.92.1 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

F.92.2 Member Function Documentation

BiometricEvaluation::Time::Watchdog::throw (Error::NotImplemented , Error::ParameterError)

Construct a new Watchdog object.
Parameters

in	*type*	The type of timer, ProcessTime or RealTime.

Exceptions

Error::NotImplemented	The type of watchdog requested is not implemented.
Error::ParameterError	The type is invalid.

Warning

Watchdog::PROCESSTIME is not supported under Cygwin.

void BiometricEvaluation::Time::Watchdog::setInterval (uint64_t *interval*)

Set the interval for the timer, but don't start the timer. Setting a value of 0 will essentially disable the timer. Timer intervals are in microseconds, however actual intervals are dependent on the resolution of the system clock, and may not be at microsecond resolution.
Parameters

in	*interval*	The timer interval, in microseconds.

void BiometricEvaluation::Time::Watchdog::start ( ) throw Error::StrategyError)

Start a watchdog timer.
Exceptions

Error::StrategyError	Could not register the signal handler, or could not create the timer.

void BiometricEvaluation::Time::Watchdog::stop ( ) throw Error::StrategyError)

Stop a watchdog timer.

Exceptions

Error::StrategyError	Could not clear the timer.

bool BiometricEvaluation::Time::Watchdog::expired ( )

Indicate whether the watchdog timer expired.

Returns

true if the timer expired, false otherwise.

void BiometricEvaluation::Time::Watchdog::setCanSigJump ( )

Indicate that the signal handler can jump into the application code after handling the signal.

void BiometricEvaluation::Time::Watchdog::clearCanSigJump ( )

Clears the flag for the Watchdog object to indicate that the signal jump block is no longer valid.

void BiometricEvaluation::Time::Watchdog::setExpired ( )

Set a flag to indicate the timer expired.

void BiometricEvaluation::Time::Watchdog::clearExpired ( )

Clear the flag indicating the timer expired.

F.92.3 Member Data Documentation

const uint8_t BiometricEvaluation::Time::Watchdog::PROCESSTIME = 0 [static]

A Watchdog based on process time.

const uint8_t BiometricEvaluation::Time::Watchdog::REALTIME = 1 [static]

A Watchdog based on real (wall clock) time.

F.93 BiometricEvaluation::Process::Worker Class Reference

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

`#include <be_process_worker.h>`

Public Member Functions

- virtual int32_t workerMain ()=0

  The method that will get called to start execution by a ProcessManager.
- tr1::shared_ptr< void > getParameter (const string &name)

  Obtain a parameter passed to this Worker.
- double getParameterAsDouble (const string &name)

  Obtain a parameter passed to this Worker as a double.
- int64_t getParameterAsInteger (const string &name)

Obtain a parameter passed to this Worker as an integer.

- string getParameterAsString (const string &name)

  Obtain a parameter passed to this Worker as a string.

- void setParameter (const string &name, tr1::shared_ptr< void > argument)

  Pass a parameter to this Worker.

- void stop ()

  Tell this Worker to return ASAP.

- void throw (Error::StrategyError)

  Perform initialization for communication from Worker to Manager.

- void throw (Error::StrategyError)

  Perform initialization for communication from Manager to Worker.

- int const throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Obtain the pipe used to send messages to this Worker.

- int const throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Obtain the pipe used to receive messages to this Worker.

- void sendMessageToManager (const Memory::uint8Array &message) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Send a message to the Manager.

- void receiveMessageFromManager (Memory::uint8Array &message) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Receive a message from the Manager.

- void throw (Error::StrategyError)

  Perform general communication initialization from Constructor.

- virtual ~Worker ()

  Worker destructor.

Protected Member Functions

- Worker ()

  Worker constructor.

- bool waitForMessage (int numSeconds=-1) const

  Block while waiting for a message from the Manager.

Protected Attributes

- bool const

  Determine if the parent has requested this child to exit.

F.93.1 Detailed Description

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

F.93.2 Member Function Documentation

virtual int32_t BiometricEvaluation::Process::Worker::workerMain ( ) [pure virtual]

The method that will get called to start execution by a ProcessManager.

Returns

Status code.

Note

If an object of this class is added to a Process::ForkManager object, the implementation of Process::-Worker::workerMain() should release all resources prior to returning.

tr1::shared_ptr<void> BiometricEvaluation::Process::Worker::getParameter (const string & *name*)

Obtain a parameter passed to this Worker.

Parameters

name	The parameter name to retrieve.

Returns

shared_ptr to the parameter argument.

Attention

If name does not exist, a new shared_ptr will be set for name.

double BiometricEvaluation::Process::Worker::getParameterAsDouble (const string & *name*)

Obtain a parameter passed to this Worker as a double.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as a double.

Attention

If name does not exist, a new shared_ptr<double> will be set for name.

int64_t BiometricEvaluation::Process::Worker::getParameterAsInteger (const string & *name*)

Obtain a parameter passed to this Worker as an integer.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as an integer.

Attention

If name does not exist, a new shared_ptr<int64_t> will be set for name.

string BiometricEvaluation::Process::Worker::getParameterAsString (const string & *name*)

Obtain a parameter passed to this Worker as a string.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as a string.

Attention

If name does not exist, a new shared_ptr<string> will be set for name.

void BiometricEvaluation::Process::Worker::setParameter (const string & *name*, tr1::shared_ptr< void > *argument*)

Pass a parameter to this Worker.

Parameters

name	A unique identifier for this parameter
argument	A shared_ptr to the object to store.

void BiometricEvaluation::Process::Worker::stop ( )

Tell this Worker to return ASAP.

Attention

This method should not be overridden.

void BiometricEvaluation::Process::Worker::throw (Error::StrategyError)

Perform initialization for communication from Worker to Manager.

Note

Behavior is undefined if called by a non-Manager.

Exceptions

Error::StrategyError	Communications not enabled.

void BiometricEvaluation::Process::Worker::throw (Error::StrategyError)

Perform initialization for communication from Manager to Worker.

Note

Behavior is undefined if called by a non-Worker.

Exceptions

Error::StrategyError	Communications not enabled.

int const BiometricEvaluation::Process::Worker::throw (Error::ObjectDoesNotExist , Error::StrategyError)

Obtain the pipe used to send messages to this Worker.

Returns

Sending pipe.

Exceptions

Error::ObjectDoesNotExist	Worker exiting soon, communication disabled.
Error::StrategyError	Communications not enabled.

int const BiometricEvaluation::Process::Worker::throw (Error::ObjectDoesNotExist , Error::StrategyError)

Obtain the pipe used to receive messages to this Worker.

Returns

Receiving pipe.

Exceptions

Error::ObjectDoesNotExist	Worker exiting soon, communication disabled.
Error::StrategyError	Communications not enabled.

void BiometricEvaluation::Process::Worker::sendMessageToManager (const Memory::uint8Array & *message*) throw Error::ObjectDoesNotExist, Error::StrategyError)

Send a message to the Manager.

Parameters

in	*message*	Message to send.

Exceptions

Error::ObjectDoesNotExist	Widowed pipe.
Error::StrategyError	Communications not enabled.

void BiometricEvaluation::Process::Worker::receiveMessageFromManager (Memory::uint8Array & *message*) throw Error::ObjectDoesNotExist, Error::StrategyError)

Receive a message from the Manager.

Parameters

out	*message*	Buffer to store the received message.

Exceptions

Error::ObjectDoesNotExist	Widowed pipe.
Error::StrategyError	Communications not enabled.

See Also

waitForMessage

void BiometricEvaluation::Process::Worker::throw (Error::StrategyError)

Perform general communication initialization from Constructor.
Exceptions

Error::StrategyError	Error in initialization.

bool BiometricEvaluation::Process::Worker::waitForMessage (int *numSeconds* = `-1`) const `[protected]`

Block while waiting for a message from the Manager.
Parameters

numSeconds	Number of seconds to wait for a message, or any value < 0 to wait forever.

Returns

true once a message is ready to be read or false if an error occured.

F.93.3 Member Data Documentation

bool BiometricEvaluation::Process::Worker::const `[protected]`

Determine if the parent has requested this child to exit.

Returns

Whether or not this child should exit.

Attention

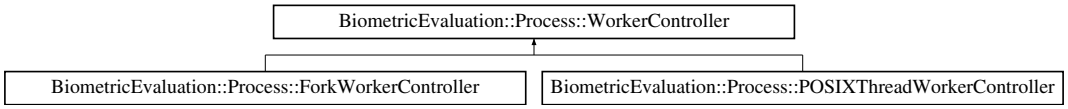
This method should not be overridden.

F.94 BiometricEvaluation::Process::WorkerController Class Reference

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

`#include <be_process_workercontroller.h>`

Inheritance diagram for BiometricEvaluation::Process::WorkerController:



Public Member Functions

- WorkerController (tr1::shared_ptr< Worker > worker)
- virtual void sendMessageToWorker (const Memory::uint8Array &message) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Send a message to the Worker contained within this WorkerController.
- virtual void setParameter (const string &name, tr1::shared_ptr< void > argument)

  Set the parameter to be passed to the Worker.
- virtual void setParameterFromDouble (const string &name, double argument)

  Set a double parameter to be passed to the Worker.
- virtual void setParameterFromInteger (const string &name, int64_t argument)

  Set an integer parameter to be passed to the Worker.
- virtual void setParameterFromString (const string &name, const string &argument)

  Set a string parameter to be passed to the Worker.
- virtual void throw (Error::ObjectExists)

  Reuse the Worker.
- virtual ~WorkerController ()

  WorkerController destructor.

Public Attributes

- virtual bool const = 0

  Obtain whether or not Worker is working.
- tr1::shared_ptr< Worker > const

  Obtain the Worker instance being wrapped.

Protected Attributes

- tr1::shared_ptr< Worker > _worker

F.94.1 Detailed Description

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

F.94.2 Constructor & Destructor Documentation

BiometricEvaluation::Process::WorkerController::WorkerController (tr1::shared_ptr< Worker > *worker*)

WorkerController constructor.

Parameters

worker	The Worker instance to wrap.

F.94.3 Member Function Documentation

virtual void BiometricEvaluation::Process::WorkerController::sendMessageToWorker (const Memory::uint8Array & *message*) throw Error::ObjectDoesNotExist, Error::StrategyError) [virtual]

Send a message to the Worker contained within this WorkerController.
Parameters

message	Message to send to the Worker.

Exceptions

Error::ObjectDoesNotExist	Worker receive pipe is closed (Worker object likely destroyed).
Error::StrategyError	Message sending failed.

virtual void BiometricEvaluation::Process::WorkerController::setParameter (const string & *name*, tr1::shared_ptr< void > *argument*) [virtual]

Set the parameter to be passed to the Worker.
Parameters

in	*name*	The name representing the argument in the Worker.
in	*argument*	The argument to be passed to the Worker.

Note

Subsequent calls to setParameter() with the same name will overwrite any exiting argument.

virtual void BiometricEvaluation::Process::WorkerController::setParameterFromDouble (const string & *name*, double *argument*) [virtual]

Set a double parameter to be passed to the Worker.
Parameters

in	*name*	The name representing the argument in the Worker.
in	*argument*	The double to be passed to the Worker.

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

virtual void BiometricEvaluation::Process::WorkerController::setParameterFromInteger (const string & *name*, int64_t *argument*) [virtual]

Set an integer parameter to be passed to the Worker.

Parameters

in	*name*	The name representing the argument in the Worker.
in	*argument*	The integer to be passed to the Worker.

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

virtual void BiometricEvaluation::Process::WorkerController::setParameterFromString (const string & *name*, const string & *argument*) [virtual]

Set a string parameter to be passed to the Worker.
Parameters

in	*name*	The name representing the argument in the Worker.
in	*argument*	The string to be passed to the Worker.

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

virtual void BiometricEvaluation::Process::WorkerController::throw (Error::ObjectExists) [virtual]

Reuse the Worker.
Exceptions

Error::ObjectExists	The previously started Worker is still running.

Reimplemented in BiometricEvaluation::Process::ForkWorkerController, and BiometricEvaluation::Process-::POSIXThreadWorkerController.

F.94.4 Member Data Documentation

virtual bool BiometricEvaluation::Process::WorkerController::const = 0

Obtain whether or not Worker is working.

Returns

Whether or not the Worker is working.

tr1::shared_ptr<Worker> BiometricEvaluation::Process::WorkerController::const

Obtain the Worker instance being wrapped.

Returns

Worker instance.

tr1::shared_ptr<Worker> BiometricEvaluation::Process::WorkerController::_worker [protected]

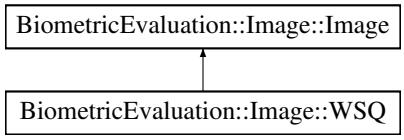
The Worker instance that is running in this child

F.95 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 > 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)

Additional Inherited Members

F.95.1 Detailed Description

A WSQ-encoded image.

F.95.2 Member Function Documentation

Memory::AutoArray<uint8_t> const BiometricEvaluation::Image::WSQ::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.

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.

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

Index