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

JULY 2, 2025

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 [23].

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

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

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

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

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

Chapter 2

Overview

The Biometric Evaluation Framework (BECommon) is a set of C++[29] classes, error codes, and design patterns used to create a common environment to provide logging, data management, error handling, and other functionality that is needed for many applications used in the testing of biometric software. The goals of the framework include:

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

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

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

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

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

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

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

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

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

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

Many packages in BECommon deal with biometric data record formats, including ANSI/NIST [6] records. In order to provide a general interface to several formats, BECommon represents the biometric data as derived from a source. For example, the `Finger` package contains classes that represent all information about a finger, including the source image and derived minutiae points. The `View` package combines the notions of a source image and derived information together into a single abstraction.

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

The `MPI` package provides wrappers around the Message Passing Interface (MPI) [21] libraries, handling all MPI communcation and error events. Many parallel applications can be greatly simplified, only implementing a few methods to process data.

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

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

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

Chapter 3

Framework

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

3.1 Versioning

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

Listing 3.1: Using the `Framework` API

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

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

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

3.2 Enumerations

As of C++ 2011, `enum` s can be strongly-typed. The Biometric Evaluation Framework makes use of these strongly-typed `enum` classes throughout. As an added convenience, functions converting to and from `enum` s, `string` s, and `int` s are defined by using a `template`, eliminating many lines of boiler-plate code and creating equivalence in functionality among `enum class` es throughout BECommon. The output stream operator << is also defined by the template.

At the core of `Framework::Enumeration` is a `const` mapping of `enum` to `string`, defined in code and instantiated at compile-time. The procedure to create a enum-to-string map is as follows:

- Include the `be_framework_enumeration.h` file to access the template definitions;
- Define the `enum` class;
- Use the `BE_FRAMEWORK_ENUMERATION_DECLARATIONS` macro to declare the enum-to-string map;

- Define the map from the `enum` elements to `std::string` objects;
- Use the `BE_FRAMEWORK_ENUMERATION_DEFINITIONS` macro to define the functions based on the map (`to_string`, etc.).

This procedure is demonstrated in Listing 3.2. The functions defined by the template exist within the `BiometricEvaluation::Framework::Enumeration` namespace. In the example application, the stream operator is used both with a call to the `to_string` function as well as directly. Typically the former where a stream operation is unavailable, calling a C program for example.

Listing 3.2: `Framework::Enumeration`

```
/*
 * color.hpp
 */
#include <be_framework_enumeration.h>
enum class Color
{
        Black,
        Blue,
        Green
};
BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
    Color, Color_EnumToStringMap);

/*
 * color.cpp
 */
#include "tfr.h"

using namespace BiometricEvaluation::Framework::Enumeration;

const std::map<Color, std::string>
Color_EnumToStringMap = {
        {Color::Black, "Black"},
        {Color::Blue, "Blue"},
        {Color::Green, "Green"}
};

BE_FRAMEWORK_ENUMERATION_DEFINITIONS(
        Color,
        Color_EnumToStringMap);

/*
 * Application
 */
#include <iostream>
int main()
{
        std::cout << to_string(Color::Black) << std::endl;
        std::cout << Color::Black << std::endl;
        std::cout << to_int_type(Color::Green) << std::endl;
        Color color = to_enum<Color>("Blue");
        std::cout << color << std::endl;
}
```

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

Chapter 4

Memory

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

4.1 AutoBuffer

The Biometric Evaluation Framework is designed to interoperate with existing C code that has its own memory management techniques, e.g. NIST Biometric Image Software [22]. 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 " [29, 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 managed memory object, or 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 data item. Attempts to insert multiple records with the same key result in an exception being thrown.

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

Listing 6.1: Using a `RecordStore`

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

    std::shared_ptr<IO::RecordStore> srs;
    try {
        srs = IO::RecordStore::createRecordStore(
            "myRecords", "My Record Store",
            IO::RecordStore::Kind::BerkeleyDB);
    } catch (Error::Exception& e) {
        cout << "Caught " << e.whatString() << endl;
        return (EXIT_FAILURE);
    }

    try {
        Memory::uint8Array theData;
        theData = getSomeData();
        srs->insert("key1", theData);

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

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

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

    // Obtain the data for all keys and write data to a file
    while (true) {
        IO::RecordStore::Record record = srs->sequence();
        cout << "Read data for key " << record.key << " of length "
            << record.data.size() << endl;
        IO::Utility::writeFile(record.data, record.key);
    }
    // The data for the key is no longer needed ...
    srs->remove("key1");
    return (EXIT_SUCCESS);
}
```

6.3 Logging

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

The logging classes in the IO package provide a straight-forward method for applications to record their progress without the need to manage the low-level storage details. Management of the log messages to the backing store is done within the Logsheet implementations. Logsheet specifies the common interface to all implementations. In addition, objects of this class can be created to provide a "Null" Logsheet where messages are not saved.

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

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

6.3.1 FileLogsheet

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

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

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

Listing 6.2: Using a FileLogsheet within a FileLogCabinet

```
1  include <iostream>
2  #include <be_io_filelogcabinet.h>
3
4  using namespace std;
5  using namespace BiometricEvaluation;
6
7  int
8  main(int argc, char* argv[]) {
9      IO::FileLogCabinet *lc;
10     try {
11         lc = new IO::FileLogCabinet("MyLogCabinet", "A Log Cabinet");
12     } catch (Error::ObjectExists &e) {
13         cout << "The Log Cabinet already exists." << endl;
14         return (EXIT_FAILURE);
15     } catch (Error::StrategyError& e) {
16         cout << "Caught " << e.whatString() << endl;
17         return (EXIT_FAILURE);
18     }
19     std::unique_ptr<IO::FileLogCabinet> alc(lc);
20     std::shared_ptr<IO::Logsheet> ls;
21     try {
22         ls = alc->newLogsheet("log01", "Log Sheet in Cabinet");
23     } catch (Error::ObjectExists &e) {
24         cout << "The log sheet already exists." << endl;
25         return (EXIT_FAILURE);
26     } catch (Error::StrategyError& e) {
27         cout << "Caught " << e.whatString() << endl;
```

```
28          return (EXIT_FAILURE);
29      }
30      ls->setAutoSync(true);  // Force write of every entry when finished
31      int i = 10;
32      *ls << "Adding an integer value " << i << " to the log." << endl;
33      ls->newEntry();
34
35      /*
36       * Further processing ....
37      */
38      return (EXIT_SUCCESS);
39  }
```

6.3.2 SysLogsheet

The `SysLogsheet` is an implementation of `Logsheet` which writes log entries to a system logger service. Objects of this class are created with a URL starting with `syslog://`. When using a system logger, the URL must give the host name of the logger as well as the network port: `syslog://node00:4315` for example. The system logger must understand the Syslog protocol as specified in RFC5424 [30].

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

6.3.3 AutoLogger

The `AutoLogger` provides the capability to automatically add entries to a `Logsheet` object on a periodic basis. The content of each log entry is returned from a callback function that is provided to the `AutoLogger` object upon construction.

Listing 6.3 shows an example code section using the `AutoLogger` class.

Listing 6.3: Using an `AutoLogger` Object

```
1  #include <chrono>
2  #include <iostream>
3  #include <be_io_autologger.h>
4  #include <be_io_filelogsheet.h>
5
6  using namespace std;
7  using namespace BiometricEvaluation;
8
9  string logEntry()
10 {
11         static int entryNum{0};
12         std::stringstream sstream{};
13         const auto tp_utc{std::chrono::system_clock::now()};
14         sstream << __FUNCTION__ << " call number "
15             << std::to_string(++entryNum) << "; date is "
16             << std::chrono::current_zone()->to_local(tp_utc);
17         return (sstream.str());
18 }
19
20 int
21 main(int argc, char *argv[])
22 {
23         std::string lsname1 = "./autologger_logsheet1.log";
24         std::shared_ptr<IO::Logsheet> logsheet1{};
```

```
25          logsheet1 = std::make_shared<IO::FileLogsheet>(
26                  "file://" + lsname1, "Autologger one sheet");
27          IO::AutoLogger logger1{};
28
29          try {
30                  cout << "Creating AutoLogger object with Logsheet: ";
31                  logger1 = IO::AutoLogger(logsheet1, &logEntry);
32          } catch (const Error::StrategyError &e) {
33                  cout << "caught " << e.what() << endl;
34                  return (EXIT_FAILURE);
35          }
36          cout << "Attempting to log asynchronously: " << flush;
37          try {
38                  logger1.setComment("Logging test");
39                  logger1.startAutoLogging(chrono::milliseconds(333));
40                  auto taskID = logger1.getTaskID();
41                  cout << "logger1 Task ID is " << taskID;
42                  sleep(30);      // Give time for the log to fill.
43                  logger1.addLogEntry();
44                  logger1.stopAutoLogging();
45          } catch (const Error::Exception &e) {
46                  cout << "Caught " << e.what() << "; failure." << endl;
47                  return (EXIT_FAILURE);
48          }
49
50          return (EXIT_SUCCESS);
51  }
```

Some representative entries in the log sheet.

```
Description: Autologger one sheet
# Autolog started. Interval: 333000 microseconds.
E 0000000001 logEntry call number 1; date is 2025-06-24 09:25:59.360389308 "test"
E 0000000002 logEntry call number 2; date is 2025-06-24 09:25:59.710192609 "test"
E 0000000003 logEntry call number 3; date is 2025-06-24 09:26:00.044639493 "test"
E 0000000004 logEntry call number 4; date is 2025-06-24 09:26:00.379022306 "test"
E 0000000005 logEntry call number 5; date is 2025-06-24 09:26:00.713566410 "test"
E 0000000006 logEntry call number 6; date is 2025-06-24 09:26:01.047245175 "test"
E 0000000007 logEntry call number 7; date is 2025-06-24 09:26:01.381046513 "test"
# Autolog stopped.
```

6.4 Properties

The `Properties` class is used to store simple key-value string pairs, and the `PropertiesFile` class stores the properties in a text file. Applications can use a `PropertiesFile` object to manage runtime settings that are persistent across invocations, or use a `Properties` object to store some settings in memory.

Listing 6.4: Using a `PropertiesFile` Object

```
1  int
2  main(int argc, char* argv[]) {
3
4
5          /* Construct a Properties object with initialization */
6          IO::PropertiesFile rwProps{"myProps", IO::Mode::ReadWrite,
```

```
            {{"One", "1"}, {"Two", "Two"}, {"Three", "3.0"}}};

        /* Set a property from an integer value */
        rwProps.setPropertyFromInteger("One", 1);

        /* Get a property as an integer value */
        try {
                int val = rwProps.getPropertyAsInteger("One");
                std::cout << "Property One is set to " << val << ".\n";
        } catch (const Error::Exception&) {
                std::cout << "Could not retrieve property.\n";
        }

        /* Set a new property */
        rwProps.setProperty("Four", "Four");

        /* Overwrite a default property */
        try {
                rwProps.setProperty("One", "New Value");
        } catch (const Error::Exception&) {
                std::cout << "Failed to overwrite a default value" << std::endl;
        }
        std::string  sval = rwProps.getProperty("One");
        std::cout << "Property One is now set to " << sval << ".\n";
}
```

6.5 Compressor

Support for data compression and decompression can be found in the Biometric Evaluation Framework through the `Compressor` class hierarchy. `Compressor` is an abstract base class defining several pure-virtual methods for compression and decompression of buffers and files. Derived classes implement these methods and can be instantiated through the factory method in the base class. As such, children should also be enumerated within `Compressor::Kind`. The Biometric Evaluation Framework comes with an example, `GZIP`, which compresses and decompresses the `gzip` format through interaction with `zlib` [8].

Listing 6.5: Using a `Compressor` Object

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

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

Listing 6.6: Setting `Compressor` Options

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

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

Chapter 7

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 7.1 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 11.3 on page 38).

Listing 7.1: 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 7.1. 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 7.2 on the following page. Changing the digest parameter to `"sha1"` would make the program emulate `` `openssl sha1` ``.

Listing 7.2: `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 8

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.

8.1 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 8.1 shows how an application can use a `Timer` object to obtain the amount of time used for the execution of a block of code.

Listing 8.1: Using the `Timer`

```
#include <be_time_timer.h>

int main(int argc, char *argv[])
{
        Time::Timer aTimer = new Time::Timer();

        try {
                aTimer->start();
                // do something useful, or not
                aTimer->stop();
                std::cout << "Elapsed time: " << aTimer->elapsed() << std::endl;
        } catch (Error::StrategyError &e) {
                std::cerr << "Failed to create timer." << std::endl;
        }
}
```

8.2 Limiting Execution Time

The `Watchdog` class allows applications to limit 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 8.2 shows how an application can use a `Watchdog` object to limit the amount of process time for a block of code.

Listing 8.2: 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 aTimer;

    BEGIN_WATCHDOG_BLOCK(theDog, watchdogblock1);
        aTimer.start();
        // Do something that may take more than 300 usecs
        aTimer.stop();
        std::cout << "Total time was " <<  aTimer.elapsed() << std::endl;
    END_WATCHDOG_BLOCK(theDog, watchdogblock1);
    if (theDog->expired()) {
        aTimer.stop();
        std::cerr << "That took too long." << std::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 8.2 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 9

Process Information and Control

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

9.1 Process Statistics

When a application is running, there may be a need to obtain information of the process executing that application. The `Process::Statistics` class 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. In addition, CPU user and system times can be gathered for all threads belonging to the process.

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 9.1 shows how an application can use the `Statistics` API.

Listing 9.1: Gathering Process Statistics

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

int main(int argc, char *argv[])
{
    Process::Statistics stats{};
    cout << "success.\n";

    uint64_t userstart, userend;
    int64_t diff;
    try {
        /*
         * Obtain the user time needed to run some code ...
         */
        std::tie(userstart, std::ignore) = stats.getCPUTimes();
        cout << "Total User time at start: " << userstart << " : ";

        // Do some long processing....

        std::tie(userend, std::ignore) = stats.getCPUTimes();
        cout << "At end: " << userend << ": ";
        diff = userend - userstart;
        cout << "User time elapsed is " << diff << endl;
```

```
25
26          /*
27           * Obtain the memory usage of the current process ...
28           */
29          uint64_t vmrss, vmsize, vmpeak, vmdata, vmstack;
30          std::tie(vmrss, vmsize, vmpeak, vmdata, vmstack) = stats.getMemorySizes();
31          cout << "\tRSS: " << vmrss;
32          cout << " : Size: " << vmsize;
33          cout << " : Peak: " << vmpeak;
34          cout << " : Data: " << vmdata;
35          cout << " : Stack: " << vmstack << endl;
36
37          /*
38           * Obtain the user and system times for all threads.
39           */
40          auto allStats = logstats->getTasksStats();
41          for (auto [tid, utime, stime]: allStats) {
42                  cout << "TID is " << tid <<
43                  " utime is " << utime <<
44                  ", stime is " << stime << '\n';
45          }
46      } catch (Error::Exception) {
47          cout << "Caught " << e.getInfo() << endl;
48      }
49
50  }
```

In addition to using the `Statistics` API to gather statistics to be returned from the function call, the API provides a means to have a set of statistics logged either synchronously or asynchronously to a pair of `Logsheet` objects. (See Section 6.3 on page 16) In addition, these sheets can be contained within a `LogCabinet`. Applications can start and stop logging at will to these sheets. Post mortem analysis can then be done on the entries in the log. Listing 9.2 on the facing page shows the use of logging.

If the `Statistics` object is constructed with a `LogCabinet` object, the log sheets will have a file name constructed from the process name (i.e. the application executable) and the process ID. Alternatively, the `Statistics` object can be constructed with one or two `Logsheet` objects, giving the application more control over where the files are stored.

An example log sheet 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
```

If the optional task statistics are to be logged, a separate file is created in the `LogCabinet` with an appropriate name. Entries in this sheet are of this form:

```
Description: Statistics for all tasks under test_be_process_statistics (PID 29193)
# Parent-ID {task-ID utime stime} ...
# Statistics auto-logger task is marked with (L)
E 0000000001 29193 {29193, 0.25, 0}
E 0000000002 29193 {29193, 0.5, 0}
E 0000000003 29193 {29193, 0.73, 0}
E 0000000004 29193 {29193, 0.96, 0}
E 0000000005 29193 {29193, 1.2, 0}
E 0000000006 29193 {29193, 1.43, 0}
```

```
# Autolog started. Interval: 1000000 microseconds.
E 0000000007 29193 {29193, 1.67, 0} {29197(L), 0, 0}
E 0000000008 29193 {29193, 1.67, 0} {29197(L), 0, 0}
E 0000000009 29193 {29193, 1.67, 0} {29197(L), 0, 0}
E 0000000010 29193 {29193, 1.67, 0} {29197(L), 0, 0}
E 0000000011 29193 {29193, 1.67, 0} {29197(L), 0, 0}
E 0000000012 29193 {29193, 1.67, 0} {29197(L), 0, 0}
E 0000000013 29193 {29193, 1.67, 0} {29197(L), 0, 0} {29198, 0, 0} {29199, 0.01, 0}
E 0000000014 29193 {29193, 1.67, 0} {29197(L), 0, 0} {29198, 0.19, 0} {29199, 0.19, 0}
E 0000000015 29193 {29193, 1.67, 0} {29197(L), 0, 0} {29198, 0.38, 0.01} {29199, 0.4, 0}
```

The above example contains entries for the single task under the process, then once auto-logging has started, more threads were created. Note that the thread responsible for logging is identified with the `(L)` tag.

When the `LogCabinet` is used, the `Statistics` object creates the `Logsheet` objects with an appropriate description and comment entry with column headers. Each gathering of the statistics results in a single log entry.

Listing 9.2: 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[])
{
    std::shared_ptr<IO::FileLogCabinet> lc;
    lc.reset(new IO::FileLogCabinet("statLogCabinet", "Cabinet for Stats"));
    std::unique_ptr<Process::Statistics> logstats;

    try {
        // Auto-log both process and task statistics
        logstats.reset(new Process::Statistics(lc, true));
    } 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;
        return (EXIT_FAILURE);
    }

    // Set up asynchronous logging, every 1/10 second
    try {
        logstats->startAutoLogging(100000);
    } catch (Error::ObjectExists &e) {
        cout << "Caught " << e.getInfo() << endl;
        return (EXIT_FAILURE);
    }

```

```
38      // Do some other work
39
40      // Stop logging
41      logstats->stopAutoLogging();
42  }
```

9.2 Process Management

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

9.2.1 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 made parallel.

9.2.2 Worker

In the application using a `Manager`, a `Worker` subclass must be implemented. An example `Worker` is shown in Listing 9.3. 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 (9.2.3) `setParameter()` method. Within the `Worker` instance, the parameters are then retrieved with `getParameter()` when provided with the unique parameter name.

A responsible worker performs its operations as fast as it can. However, at any given time, the manager may ask the worker to stop. It then becomes the *responsibility of the worker* to stop as soon as possible. The `Worker` is notified of the stop request through its `stopRequested()` method. Note that the manager does **not** force the worker to stop, though prolonged work or cleanup in the worker would likely produce undesired results in the client application. As such, a responsible worker checkpoints itself to prepare for premature stops requested by the manager. While it is important for a worker to stop as soon as possible after the request is received, it is also important not to leave work in an unsynchronized state. In Listing 9.3, 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 9.3: A Responsible `Worker` Implementation

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

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

After a manager starts its workers, the manager has the option of waiting until all `Worker` s exit `workerMain()` before continuing code execution. If not waiting, there are several methods the manager can perform to keep track of the status of the workers. Even if not waiting for workers to return, a responsible manager will wait a reasonable amount of time for workers to `return` before application termination. An example of this reasonable waiting period can be seen in Listing 9.4 on the next page.

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

Listing 9.4 is a continuation of Listing 9.3 on page 30 demonstrating the use of `Manager` s and `Worker Controller` s.

Listing 9.4: Using `Manager` s and `WorkerController` s

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

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

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

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

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

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

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

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

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

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

9.2.4 Communications

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

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

Listing 9.5 and Listing 9.6 are continuations of Listing 9.3 on page 30 and Listing 9.4 on the preceding page respectively, showing an example of communication, using `std::string` messages.

Listing 9.5: `Worker` Communication

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

Listing 9.6: `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 10

System

The `System` package provides a set of functions 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, and can be used to dynamically tailor the application behavior, or simply to provide additional information in a runtime log.

Listing 10.1 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 10.1: 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();
        uint64_t vmSize;
        Process::Statistics stats{};
        std::tie(std::ignore, vmSize, std::ignore, std::ignore, std::ignore)
            = stats.getMemorySizes();
        memSize -= vmSize;   // subtract off memory used by parent

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

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

10.0.1 MemoryLogger

The `MemoryLogger` is used to periodically add a log entry to an existing `Logsheet`, each entry containing system memory usage. Upon construction logging will be off and must be started by the application, specifying the logging interval.

Listing 10.2: Using the `System::MemoryLogger` Class

```
#include <chrono>
#include <iostream>

#include <be_io_filelogsheet.h>
#include <be_io_utility.h>
#include <be_system_memlog.h>

namespace BE = BiometricEvaluation;

int
main(int argc, char *argv[])
{
        auto logsheetPath = BE::IO::Utility::createTemporaryFile("memlog");
        std::shared_ptr<BE::IO::Logsheet> logsheet =
            std::make_shared<BE::IO::FileLogsheet>("file://" + logsheetPath);
        BE::System::MemoryLogger memlog{logsheet};

        std::cout << "Starting autolog... " << std::flush;
        const auto interval = std::chrono::seconds(2);
        try {
                memlog.startAutoLogging(interval, true);
        } catch (const std::exception &e) {
                std::cerr << e.what() << '\n';
                return (EXIT_FAILURE);
        }

        //
        // Do some memory intensive work ...
        //
        std::cout << "Stopping autolog... " << std::flush;
        try {
                memlog.stopAutoLogging();
        } catch (const std::exception &e) {
                std::cerr << e.what() << '\n';
                return (EXIT_FAILURE);
        }
        std::cout << " [OKAY]\n";

        return (EXIT_SUCCESS);
}
```

Chapter 11

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.

11.1 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 11.2). Data types in the Image namespace do not perform any translation of scale units or sizing, as each set of attributes is copied directly from the image data itself when possible.

The same applies to images encapsulated in biometric records. Although some biometric records have fields for image attributes like dimensions and resolution, the corresponding fields of an `Image` class are **not** populated with their contents. The `Image` namespace data types *are* used outside of the namespace, such as in finger views, to retrieve image attributes stored as part of the biometric record. Applications can compare those values against the values within the `Image` object, as in most cases those values are taken directly from the underlying image data. See Chapter 15 on page 51 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.

11.2 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.

11.3 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.

11.4 JPEG

The `JPEG` class represents an image encoded according to the JPEG image standard [16]. Decompression and grayscale conversion are accomplished via `libjpeg` [14].

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.

11.5 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` [22].

11.6 JPEG2000

The `JPEG2000` class provides Image class functionality to JPEG 2000-encoded images [15]. 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`) [19]. `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.

11.7 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.

11.8 PNG

The PNG class represents an image encoded according to the PNG image standard [11]. Decompression is provided by libpng [26].

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.

11.9 TIFF

The TIFF provides the ability to decompress many TIFF-encoded images. Decompression routines are provided by libtiff [27]. Like most other Image classes, only basic grayscale and RGB-based images are parsable. The TIFF class will throw a NotImplemented exception in the event that unsupported TIFF data is provided.

11.10 WSQ

Images encoded in the WSQ-image standard [31] are represented by the WSQ class. The WSQ decompressor found in NIST Biometric Image Software [22], 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 thrown as exceptions.

11.11 BMP

The bitmap image file format [20] is decoded by the BMP class. Only images with the 40-byte BITMAPINFOHEADER, uncompressed or RLE8 compression are supported. The bits-per-pixel value can be 8, 24, or 32.

Chapter 12

Video

The `Video` package is used to access video (and, in the future, audio) streams from containers in several formats, such as MPEG4. The classes in this package rely on the FFmpeg [12] libraries to de-multiplex video streams from a container, and to decode the streams and retrieve the frames from the video.

12.1 Container

`Container` objects can be instantiated in three ways:

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

By careful coding, the application can prevent duplicate copies of the container buffer when using method three. By taking advantage of C++ 2011 move semantics, BECommon and the application avoid duplicate copies. See Listing 12.1 for examples of using all three methods.

12.2 Stream

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

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

Listing 12.1 shows the use of `Container` and `Stream`.

Listing 12.1: Using the Video Framework

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

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

```
62          }
63          /*
64           * Read all the frames, one at a time, scaled down and converted
65           * to 8-bit grayscale.
66           */
67          float scaleFactor = 0.5;
68          Image::PixelFormat pixelFormat = Image::PixelFormat::Gray8;
69          stream->setFrameScale(scaleFactor, scaleFactor);
70          stream->setFramePixelFormat(pixelFormat);
71          uint64_t expectedCount = stream->getFrameCount();
72
73          cout << "First video stream: " << stream->getFPS() << " FPS, "
74              << expectedCount << " frames." << endl;
75          /*
76           * The frame count can be zero, meaning unknown. If that is the case,
77           * loop until a parameter error is indicated.
78           */
79          if (expectedCount == 0)
80                  expectedCount = 99999999;
81          uint64_t count = 0;
82          for (uint64_t f = 1; f <= expectedCount; f++) {
83                  try {
84                          auto frame = stream->getFrame(f);
85                          count++;
86                          /* Do something with frame.data */
87                          std::cout << "frame size is "
88                              << frame.size.xSize << "x" << frame.size.ySize
89                              << std::endl;
90                  } catch (Error::ParameterError &e) {
91                          cout << "No more frames.";
92                          break;
93                  } catch (Error::Exception &e) {
94                          std::cout << "Caught " << e.whatString() << endl;
95                          return (EXIT_FAILURE);
96                  }
97          }
98          cout << "Retrieved " << count << " frames." << endl;
99          return (EXIT_SUCCESS);
100 }
```

Chapter 13

Device

The `Device` package consists of classes, constants, and other structures used to communicate with hardware devices. These include smartcards that conforms to the ISO Smartcard standard [5].

13.1 TLV

The `TLV` class represents a single tag-length-value object as described in [5]. The data for a TLV can be represented in two manners:

- As a "raw" set of octets; this is the format used by smartcards;
- As an object giving accessed to the parsed fields, data, and children.

Both "constructed" and "primitive" basic-encoding-rule (BER) TLV objects are supported by the `TLV` class. Methods are provided to obtain the children of a constructed BER-TLV and to obtain the data of a primitive BER-TLV.

13.2 Smartcard

13.2.1 APDU

The `APDU` represents an Application Protocol Data Unit (APDU) that is sent to a card. An APDU object directly represents the data according to [5] as all fields of the the class are public. Applications can send an APDU to the card, but the more effective approach is to subclass `Smartcard` and wrap APDU communication with methods that are specific to the type of card.

13.2.2 Smartcard Communication

The `Smartcard` class provides generic access to a any card that is inserted in the system. An application on the card can be activated during construction. Card data objects can be retrieved based on the object ID, and any APDU can be sent to the card.

Because communicating with a card depends on a command/response protocol, `Smartcard` provides methods to retrieve the response returned by the card. This retrieval is useful when the status words must be examined as many commands can result in several values for each status word.

Listing 13.1: Accessing a PIV smartcard

```
#include <iostream>
#include <be_device_smartcard.h>
#include <be_device_tlv.h>
#include <be_error_exception.h>

int main(int argc, char *argv[])
{
        std::cout << "Attempt to activate PIV: " << std::endl;
        for (int i = 0; i < 4; i++) {
                try {
                        std::cout << "\tReader " << i << ": ";
                        BE::Device::Smartcard smc(i,
                             {0xA0, 0x00, 0x00, 0x03, 0x08, 0x00, 0x00,
                              0x10, 0x00, 0x01, 0x00});
                        std::cout << "Found." << std::endl;

                        std::cout << "Get Card Capability Container: "
                            << std::endl;;
                        BE::Memory::uint8Array
                            objID{0x5C, 0x03, 0x5F, 0xC1, 0x07};
                        auto obj = smc.getDedicatedFileObject(objID);

                        /* The CCC is contained within a TLV */
                        std::cout << BE::Device::TLV::stringFromTLV(obj, 1);

                        /* Do something with the TLV data, which is the CCC */
                        BE::Device::TLV tlv(obj);
                        processCCC(tlv.getPrimitive());

                // The card responded with something other than normal
                // processing complete, catch the exception from the
                // Framework so the status words can be examined.
                } catch (BE::Device::Smartcard::APDUException &e) {
                                std::cout << "Bad response: ";
                                printf("0x%02hhX%02hhX\n",
                                    e.response.sw1, e.response.sw2);
                                std::cout << "Sent APDU: " << std::endl;
                                // Dump the octets from the sent APDU
                                dumpUint8Array(e.apdu);
                } catch (BE::Error::ParameterError &e) {
                                std::cout << "Caught " << e.whatString();
                } catch (BE::Error::StrategyError &e) {
                        std::cout << "Other error: " << e.whatString();
                }
                std::cout << std::endl;
        }
        return (EXIT_SUCCESS);
}
```

The example code in Listing 13.1 shows how to activate the PIV smartcard and retrieve one of its data objects.

Chapter 14

Feature

The Feature package contains those items that relate to the representation of biometric features, such as fingerprint minutiae, facial features (eyes, etc.), and related information. Objects of these class types are typically associated with View (Chapter 15 on page 51) or DataInterchange (Chapter 19 on page 63) 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.

14.1 ANSI/NIST Features

The ANSI/NIST [6] 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 16 on page 53).

See Listing 16.1 on page 54 for a complete example of how to obtain the fingerprint minutiae data from an ANSI/NIST record. If only extended feature set data is required from the file, a Feature::AN2K11EFS::ExtendedFeatureS object can be created directly from the file or memory buffer.

14.1.1 ANSI/NIST 2011 Extended Feature Sets

The 2011 edition of the ANSI/NIST standard [7] adds a new form of feature data representation to the Type-9 fingerprint minutiae record. The extended feature set information is represented by an object that can be retrieved from the AN2KMinutiaeDataRecord object created from the data file.

Listing 14.1 shows how to read the extended feature set data from an ANSI/NIST file, both as a data interchange object (see Section 19 on page 63) or an extended feature set object constructed directly from a file.

Listing 14.1: Using AN2K Extended Feature Sets

```
1 #include <iostream>
2 #include <be_data_interchange_an2k.h>
3 #include <be_feature_an2k11efs.h>
4
```

```
5  /*
6   * This test program exercises the Evaluation framework to process AN2K
7   * records stored in a RecordStore. The intent is to model what a real
8   * program would do by retrieving AN2K records, doing some processing
9   * on the image, and displaying the results.
10  */
11 using namespace BiometricEvaluation;
12
13 static void
14 printAN2K11EFS(Feature::AN2K11EFS::ExtendedFeatureSet &efs)
15 {
16         Image::ROI roi = efs.getImageInfo().roi;
17         std::cout << "ROI:\n"
18             << "\tSize: ("
19             << roi.size.xSize << "," << roi.size.ySize << ")\n"
20             << "\tOffset: ("
21             << roi.horzOffset << "," << roi.vertOffset << ")\n"
22             << "\tPath: ";
23         for (auto const& point: roi.path) {
24                 std::cout << point << " ";
25         }
26         std::cout << "\n";
27
28         std::cout << "Image Info:\n" << efs.getImageInfo() << "\n\n";
29
30         Feature::AN2K11EFS::CorePointSet cps = efs.getCPS();
31         std::cout << "CPS: Have " << cps.size() << " EFS core point(s):\n";
32         for (auto const& cp: cps) {
33                 std::cout << "\t" << cp << "\n";
34         }
35
36         Feature::AN2K11EFS::DeltaPointSet dps = efs.getDPS();
37         std::cout << "DPS: Have " << dps.size() << " EFS delta point(s):\n";
38         for (auto const& dp: dps) {
39                 std::cout << "\t" << dp << "\n";
40         }
41
42         Feature::AN2K11EFS::MinutiaPointSet mps = efs.getMPS();
43         std::cout << "MPS: Have " << mps.size() << " EFS minutia point(s):\n";
44         for (auto const& mp: mps) {
45                 std::cout << mp << "\n";
46         }
47
48         std::cout << "No Features Present:\n";
49         std::cout << efs.getNFP();
50
51         std::cout << "\nMinutiae Ridge Count Information:\n";
52         auto mrci = efs.getMRCI();
53         std::cout << mrci << "\n";
54 }
55
56 int
57 main(int argc, char* argv[]) {
58
59         std::string fname = "test_data/type9-efs.an2k";
60         /*
```

```
61          * Read the EFS data from the DataInterchange::AN2KRecord object
62          */
63         std::cout << "Extended Feature Set data in " << fname << ": ";
64         try {
65                 DataInterchange::AN2KRecord an2k(fname);
66                 std::vector<Finger::AN2KMinutiaeDataRecord> minutiae =
67                     an2k.getMinutiaeDataRecordSet();
68                 printAN2K11EFS(*minutiae[0].getAN2K11EFS());
69         } catch (Error::Exception &e) {
70                 std::cout << "Failed; caught " << e.whatString() << "\n";
71         }
72
73         /*
74          * Read the EFS data by constructing directly from the filename
75          */
76         try {
77                 Feature::AN2K11EFS::ExtendedFeatureSet efs(fname, 1);
78                 printAN2K11EFS(efs);
79         } catch (Error::Exception &e) {
80                 std::cout << "Failed; caught " << e.whatString() << "\n";
81         }
82 }
```

14.2 ISO/INCITS Features

The ISO [4] 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 16.2 on page 55 shows how to create a view object for the fingerprint minutiae record contained in a file.

Chapter 15

View

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

In the case where a raw image is part of the biometric record, the `View` object's related `Image` (Chapter 11 on page 37 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.

Views 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 16 on page 53), represent the specific type of biometric sample.

`View` objects are created with information taken from a biometric data record, an ANSI/NIST 2007 file, for example. Most record formats contain information about the image itself, such as the resolution and size. The 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.

Listing 15.1 shows a function that will print the information obtained from any `View` object.

Listing 15.1: View::View Class

```
1  void
2  printViewInfo(BiometricEvaluation::View::View &view)
3  {
4          cout << "Image size is " << view.getImageSize() << endl;
5          cout << "Image resolution is " << view.getImageResolution() << endl;
6          cout << "Scan resolution is " << view.getScanResolution() << endl;
7          cout << "Image color depth is " << view.getImageColorDepth() << endl;
8          cout << "Compression is " << view.getCompressionAlgorithm() << endl;
9          try {
10                 auto theImage = view.getImage();
11                 cout << "Information from the Image data item:" << endl;
12                 cout << "\tResolution: " << theImage->getResolution() << endl;
13                 cout << "\tDimensions: " << theImage->getDimensions() << endl;
14                 cout << "\tDepth: " << theImage->getColorDepth() << endl;
15         } catch (Error::Exception &e) {
16                 cout << "Caught " << e.what() << endl;
17         }
```

```
18 }
```

15.1 ANSI/NIST Views

The ANSI/NIST standard [6] describes fixed and variable resolution finger, latent, and palm image records. These are represented within BECommon by `View::AN2KView` (subclass of `View::View` and `View::AN2KViewVariableResolution`, subclass of `AN2KView`. As these classes only define the common interface for the ANSI/NIST records, objects of these class types cannot be created. These classes are further extended by classes in the `Finger`, `Latent`, and `Palm` name spaces. See 16.1 and 17.1.

Chapter 16

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 19 on page 63) 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 [6]). 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.)

16.1 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 16 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.

16.1.1 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 16.1 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 16.1: Using an AN2K Finger View

```
#include <iostream>

#include <be_finger_an2kview_fixedres.h>
#include <be_error_exception.h>
#include <be_io_utility.h>

using namespace BiometricEvaluation;
using namespace BiometricEvaluation::Framework::Enumeration;

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

        /*
         * Call the constructor that will open an existing AN2K file.
         */
        std::unique_ptr<Finger::AN2KViewFixedResolution> an2kv;
        try {
                an2kv.reset(new Finger::AN2KViewFixedResolution(
                    "test_data/type3.an2k",
                    View::AN2KView::RecordType::Type_3, 1));
        } catch (Error::DataError &e) {
                std::cout << "Caught " << e.what()  << std::endl;
                return (EXIT_FAILURE);
        } catch (Error::FileError& e) {
                std::cout << "A file error occurred: " << e.what() << std::endl;
                return (EXIT_FAILURE);
        }
        std::cout << "Image resolution is "
             << an2kv->getImageResolution() << std::endl;
        std::cout << "Image size is " << an2kv->getImageSize() << std::endl;
        std::cout << "Image color depth is "
            << an2kv->getImageColorDepth() << std::endl;
        std::cout << "Compression is " <<
            to_string(an2kv->getCompressionAlgorithm()) << std::endl;
        std::cout << "Scan resolution is "
            << an2kv->getScanResolution() << std::endl;
        std::cout << "Impression Type: " <<
            to_string(an2kv->getImpressionType()) << std::endl;

        /*
         * Get the compressed image data and process
         */
        std::shared_ptr<Image::Image> img = an2kv->getImage();
        if (img.get() == nullptr) {
                std::cout << "Image was nullptr" << std::endl;
```

```
46          } else {
47                  // Process the image data
48          }
49          /*
50           * Get the raw image data and save to a file
51           */
52          std::ofstream img_out("imgdata.raw", std::ofstream::binary);
53          Memory::uint8Array imgData{img->getRawData()};
54          img_out.write((char *)&(imgData[0]), imgData.size());
55          if (img_out.good()) {
56                  img_out.close();
57          } else {
58                  std::cout << "Error occurred when writing." << std::endl;
59          }
60          /*
61           * Get all the positions from the data record.
62           */
63          Finger::PositionSet positions = an2kv->getPositions();
64          std::cout << "There are " << positions.size() << " positions:"
65              << std::endl;
66          for (auto p: positions) {
67                  std::cout << "\t" << to_string(p) << std::endl;
68          }
69          /*
70           * Get the minutiae data records and print the minutiae points in
71           * each data record
72           */
73          auto mdrs = an2kv->getMinutiaeDataRecordSet();  // The set of records
74          std::cout << "There are " << mdrs.size() << " minutiae data records."
75              << std::endl;
76          for (auto mdr: mdrs) {
77                  for (auto mp: mdr.getAN2K7Minutiae()->getMinutiaPoints()) {
78                          std::cout << mp << std::endl;
79                  }
80          }
81
82          return(EXIT_SUCCESS);
83  }
```

16.1.2 ISO/INCITS Finger Views

The ISO [18] and INCITS [17] 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.

*(**NOTE:** Reading data from finger image records is not currently supported)*

Listing 16.2 shows how an application can create a view from an ANSI/INCTIS 378 finger minutiae format record [1].

Listing 16.2: Using an INCITS Finger View

```
1  #include <iostream>
2  #include <be_finger_ansi2004view.h>
3  #include <be_feature_incitsminutiae.h>
```

```
4  using namespace std;
5  using namespace BiometricEvaluation;
6  using namespace BiometricEvaluation::Framework::Enumeration;
7
8  int
9  main(int argc, char* argv[])
10 {
11     Finger::ANSI2004View fngv;
12     try {
13         fngv = Finger::ANSI2004View("test_data/fmr.ansi2004", "", 3);
14     } catch (Error::Exception &e) {
15         cerr << "Caught " << e.whatString()  << endl;
16         return (EXIT_FAILURE);
17     }
18     cout << "Image resolution is " << fngv.getImageResolution() << endl;
19     cout << "Image size is " << fngv.getImageSize() << endl;
20     cout << "Image color depth is " << fngv.getImageColorDepth() << endl;
21     cout << "Compression is " << fngv.getCompressionAlgorithm() << endl;
22     cout << "Scan resolution is " << fngv.getScanResolution() << endl;
23
24     Feature::INCITSMinutiae fmd = fngv.getMinutiaeData();
25     cout << "Minutiae format is " << fmd.getFormat() << endl;
26     Feature::MinutiaPointSet mps = fmd.getMinutiaPoints();
27     cout << "There are " << mps.size() << " minutiae points:" << endl;
28     for (auto mp: mps)
29         cout << mp;
30
31     Feature::RidgeCountItemSet rcis = fmd.getRidgeCountItems();
32     cout << "There are " << rcis.size() << " ridge count items:" << endl;
33     for (auto rci: rcis)
34         cout << "\t" << rci;
35
36     Feature::CorePointSet cores = fmd.getCores();
37     cout << "There are " << cores.size() << " cores:" << endl;
38     for (auto core: cores)
39         cout << "\t" << core;
40
41     Feature::DeltaPointSet deltas = fmd.getDeltas();
42     cout << "There are " << deltas.size() << " deltas:" << endl;
43     for (auto delta: deltas)
44         cout << "\t" << delta;
45
46     exit (EXIT_SUCCESS);
47 }
```

Chapter 17

Palm

The `Palm` package provides access to palm print information stored in standard record formats. Within this package are defined the common elements relevant to palm images, such as position and minutiae data.

17.1 ANSI/NIST Palm Views

The `Palm::AN2KView` class, extends `View::AN2KViewVariableResolution` (See 15) by adding methods to retrieve palm information from an ANSI/NIST ([7]) Type-15 record.

Listing 17.1 shows how an application can query the information from an ANSI/NIST data file.

Listing 17.1: Using the Palm::AN2KView Class

```
#include <iostream>
#include <be_io_utility.h>
#include <be_palm_an2kview.h>

using namespace std;
using namespace BiometricEvaluation;
using namespace BiometricEvaluation::Framework::Enumeration;

static void
printViewInfo(const Palm::AN2KView &an2kv) {
        cout << "Source Agency: " << an2kv.getSourceAgency() << endl;
        cout << "Capture Date: " << an2kv.getCaptureDate() << endl;
        cout << "Comment: [" << an2kv.getComment() << "]" << endl;

        cout << "Image resolution: " << an2kv.getImageResolution() << endl;
        cout << "Image size: " << an2kv.getImageSize() << endl;
        cout << "Image color depth: " << an2kv.getImageColorDepth() << endl;
        cout << "Compression: " << an2kv.getCompressionAlgorithm() << endl;
        cout << "Scan resolution: " << an2kv.getScanResolution() << endl;
        cout << "Impression Type: " << an2kv.getImpressionType() << endl;
        cout << "Position: " << an2kv.getPosition() << endl;
        auto qms = an2kv.getPalmQualityMetric();
        cout << "Palm Quality has " << qms.size() << " entries:" << endl;
        for (auto &qm: qms) {
                cout << "\t" << qm << endl;
        }
        shared_ptr<Image::Image> img = an2kv.getImage();
        if (img != nullptr) {
```

```
29                 cout << "Image info:" << endl;
30                 cout << "\tCompression: " << img->getCompressionAlgorithm()
31                     << endl;
32                 cout << "\tDimensions: " << img->getDimensions() << endl;
33                 cout << "\tResolution: " << img->getResolution() << endl;
34                 cout << "\tDepth: " << img->getColorDepth() << endl;
35         } else {
36                 cout << "No Image available." << endl;
37         }
38
39 }
40
41 int
42 main(int argc, char* argv[]) {
43
44         /*
45          * Call the constructor that will open an existing AN2K file.
46          */
47         std::shared_ptr<Palm::AN2KView> an2kv;
48         try {
49                 an2kv.reset(new Palm::AN2KView(
50                     "test_data/type9-15.an2k", 1));
51         } catch (Error::Exception &e) {
52                 cout << "Caught " << e.what()  << endl;
53                 return (EXIT_FAILURE);
54         }
55         printViewInfo(*an2kv);
56
57         cout << "Get the set of minutiae data records: ";
58         auto minutiae = an2kv->getMinutiaeDataRecordSet();
59         cout << "There are " << minutiae.size()
60             << " minutiae data record sets." << endl;
61         if (minutiae.size() != 0) {
62                 cout << "Minutiae Points:\n";
63                 for (auto m:
64                      minutiae[0].getAN2K7Minutiae()->getMinutiaPoints()) {
65                         cout << m << endl;
66                 }
67                 cout << "Cores:\n";
68                 for (auto c:
69                      minutiae[0].getAN2K7Minutiae()->getCores()) {
70                         cout << c << endl;
71                 }
72                 cout << "Deltas:\n";
73                 for (auto d:
74                      minutiae[0].getAN2K7Minutiae()->getDeltas()) {
75                         cout << d << endl;
76                 }
77         }
78         return(EXIT_SUCCESS);
79 }
```

Chapter 18

Face

The `Face` package provides access to facial information stored in standard record formats. Within this package are defined the common elements relevant to facial images, such as hair color, expression, pose angle, and others.

18.0.1 ISO/INCITS Face Views

The `Face::INCITSView` class, extends `View::View` (See 15) by adding methods to retrieve facial information. A `Face::INCITSView` object cannot be constructed by applications but rather this class is subclassed to represent each standard format. For example, the `ISO2005View` class represents the ISO/IEC 19794-5 [3] standard.

Listing 18.1 shows how an application can query the information from a standard ISO/INCITS-385 facial information record.

Listing 18.1: Using the Face::ISO2005View Class

```
#include <iostream>
#include <iomanip>
#include <be_face_iso2005view.h>

using namespace std;
using namespace BiometricEvaluation;
using namespace BiometricEvaluation::Framework::Enumeration;

void
printViewInfo(View::View &view)
{
        /*
         * Provided by the View::View interface.
         */
        cout << "Image resolution is " << view.getImageResolution() << endl;
        cout << "Scan resolution is " << view.getScanResolution() << endl;
        cout << "Image size is " << view.getImageSize() << endl;
        cout << "Image depth is " << view.getImageColorDepth() << endl;
        cout << "Compression is " <<
            view.getCompressionAlgorithm() << endl;

        try {
                std::shared_ptr<Image::Image> theImage = view.getImage();
                cout << "Information from the Image data item:" << endl;
                cout << "\tResolution: " << theImage->getResolution() << endl;
```

```
26                  cout << "\tDimensions: " << theImage->getDimensions() << endl;
27                  cout << "\tDepth: " << theImage->getColorDepth() << endl;
28          } catch (Error::Exception &e) {
29                  cout << "Caught " << e.what() << endl;
30          }
31          cout << "----------------------------------------" << endl;
32  }
33
34  void
35  printFaceInfo(Face::ISO2005View &facev)
36  {
37          /*
38           * Provided by the Face::INCITSView interface.
39           */
40          cout << "Gender: " << facev.getGender() << endl;
41          cout << "Eye Color: " << facev.getEyeColor() << endl;
42          cout << "Hair Color: " << facev.getHairColor() << endl;
43          cout << "Expression: " << facev.getExpression() << endl;
44
45          Face::PoseAngle pa =  facev.getPoseAngle();
46          cout << "Pose angle info: ";
47          cout << "Yaw/Uncer: " << (int)pa.yaw << "/" << (int)pa.yawUncertainty;
48          cout << "; Pitch/Uncer: "
49              << (int)pa.pitch << "/" << (int)pa.pitchUncertainty;
50          cout << "; Roll/Uncer: "
51              << (int)pa.roll << "/" << (int)pa.rollUncertainty << endl;
52
53          cout << "Image type is " << facev.getImageType() << endl;
54          cout << "Image data type is " << facev.getImageDataType()
55              << endl;
56          cout << "Color space is " << facev.getColorSpace() << endl;
57          cout << "Source type is " << facev.getSourceType() << endl;
58          cout << "Device type is " << "0x" << hex << setw(4) << setfill('0')
59              << facev.getDeviceType() << dec << endl;
60
61          Face::PropertySet properties;
62          bool haveProps = facev.propertiesConsidered();
63          if (haveProps) {
64                  facev.getPropertySet(properties);
65                  cout << "There are " << properties.size() << " properties: ";
66                  for (size_t i = 0; i < properties.size(); i++) {
67                          if (i != properties.size() - 1)
68                                  cout << properties[i] << ", ";
69                          else
70                                  cout << properties[i];
71                  }
72                  cout << endl;
73          } else {
74                  cout << "There are no properties." << endl;
75          }
76
77          Feature::MPEGFacePointSet fps;
78          facev.getFeaturePointSet(fps);
79          cout << "There are " << fps.size() << " feature points." << endl;
80          if (fps.size() != 0) {
81                  cout << "\tType\tCode\tPosition" << endl;
```

```
82          }
83          for (size_t i = 0; i < fps.size(); i++) {
84                  cout << "\t" << (int)fps[i].type
85                      << "\t" << (int)fps[i].major << "." << (int)fps[i].minor
86                      << "\t" << fps[i].coordinate
87                      << endl;
88          }
89          cout << "----------------------------------------" << endl;
90  }
91
92  int
93  main(int argc, char* argv[])
94  {
95          Face::ISO2005View facev;
96          try {
97                  facev = Face::ISO2005View("test_data/face01.iso2005", 1);
98          } catch (Error::Exception &e) {
99                  cout << "Caught " << e.what()  << endl;
100                 return (EXIT_FAILURE);
101         }
102         printViewInfo(facev);
103         printFaceInfo(facev);
104         return(EXIT_SUCCESS);
105 }
```

Chapter 19

Data Interchange

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

The design of classes in the `DataInterchange` package allows applications to create a single object from a biometric record, such as an ANSI/NIST file. After creating this object, the application can retrieve the needed information (such as finger views Chapter 16 on page 53) 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.

19.1 ANSI/NIST Data Records

The ANSI/NIST Data Interchange package contains the classes used to represent ANSI/NIST [6] 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 19.1 shows how an application can retrieve all finger latents (Type-13) and captures (Type-14) from an ANSI/NIST record. Also shown is the general record information such as the capture date, etc. Once the views are retrieved, the application obtains the set of minutiae records associated with that view. In addition, the example shows how the entire set of minutiae records can be read independent of a view.

Listing 14.1 on page 47 shows how to retrieve the extended feature set data by constructing a data interchange object.

Listing 19.1: ANSI/NIST Data Interchange

```
1 #include <iostream>
2 #include <be_data_interchange_an2k.h>
3
4 /*
5  * This test program exercises the Evaluation framework to process an AN2K
6  * records stored in a file. The intent is to model what a real program
7  * would do by retrieving AN2K records, doing some processing on the image,
```

```
 * and displaying the results.
 */
using namespace std;
using namespace BiometricEvaluation;
using namespace BiometricEvaluation::Framework::Enumeration;

static void
printRecordInfo(const DataInterchange::AN2KRecord &an2k)
{
        cout << "\tVersion: " << an2k.getVersionNumber() << endl;
        cout << "\tDate: " << an2k.getDate() << endl;
        cout << "\tDestination Agency: " <<
            an2k.getDestinationAgency() << endl;
        cout << "\tOriginating Agency: " <<
            an2k.getOriginatingAgency() << endl;
        cout << "\tTransaction Control Number: " <<
            an2k.getTransactionControlNumber() << endl;
        cout << "\tNative Scanning Resolution: " <<
            an2k.getNativeScanningResolution() << endl;
        cout << "\tNominal Transmitting Resolution: " <<
            an2k.getNominalTransmittingResolution() << endl;
        cout << "\tCapture Count: " << an2k.getFingerCaptureCount() << endl;
        cout << "\tLatent Count: " << an2k.getFingerLatentCount() << endl;
}

static void
printViewInfo(const View::AN2KViewVariableResolution &an2kv)
{
        cout << "\tRecord Type: " <<
            static_cast<std::underlying_type<
            View::AN2KView::RecordType>::type>(an2kv.getRecordType()) << endl;
        cout << "\tImage resolution: " << an2kv.getImageResolution() << endl;
        cout << "\tImage size: " << an2kv.getImageSize() << endl;
        cout << "\tImage color depth: " << an2kv.getImageColorDepth() << endl;
        cout << "\tCompression: " <<
            to_string(an2kv.getCompressionAlgorithm()) << endl;
        cout << "\tScan resolution: " << an2kv.getScanResolution() << endl;
        cout << "\tImpression Type: " << to_string(an2kv.getImpressionType()) <<
            endl;
        cout << "\tSource Agency: " << an2kv.getSourceAgency() << endl;
        cout << "\tCapture Date: " << an2kv.getCaptureDate() << endl;
        cout << "\tComment: [" << an2kv.getComment() << "]" << endl;

        /*
         * Get the image data.
         */
        auto img = an2kv.getImage();
        if (img != nullptr) {
                /* Do something with the image info and data */
                ;
        } else {
                cout << "No Image available.\n";
        }

        /*
         * Print info for the minutiae associated with this view.
```

```
64          */
65          auto minutiae = an2kv.getMinutiaeDataRecordSet();
66          cout << "\tThere are " << minutiae.size() <<
67              " minutiae data records.\n";
68  }
69
70  int
71  main(int argc, char* argv[]) {
72          try {
73                  DataInterchange::AN2KRecord an2k("test_data/a002.an2");
74                  printRecordInfo(an2k);
75                  /*
76                   * Obtain the finger capture and latent views from the
77                   * AN2k file.
78                   */
79                  int i = 0;
80                  for (auto c: an2k.getFingerCaptures()) {
81                          cout << "[Capture View " << i++ <<"]\n";
82                          printViewInfo(c);
83                          cout << "\tPosition: " << c.getPosition()
84                              << endl;
85                          cout << "[End of Capture View]\n";
86                  }
87                  i = 0;
88                  for (auto l: an2k.getFingerLatents()) {
89                          cout << "[Latent View " << i++ <<"]\n";
90                          printViewInfo(l);
91                          cout << "\tPositions: ";
92                          for (auto p: l.getPositions()) {
93                                  cout << p << " ";
94                          }
95                          cout << endl << "[End of Latent View]\n";
96                  }
97                  /*
98                   * Obtain the entire set of minutiae records from the
99                   * AN2k file, independently of the view.
100                  */
101                 auto minutiae = an2k.getMinutiaeDataRecordSet();
102                 cout << "There is a total of " << minutiae.size()
103                     << " minutiae data records in the AN2K file.\n";
104                 cout << ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n";
105         } catch (Error::Exception &e) {
106                 cout << "Failed sequence: " << e.what() << endl;
107                 return (EXIT_FAILURE);
108         }
109 }
```

19.2 INCITS Data Records

The 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 [17]. These formats include the ANSI-2004 Finger Minutiae Record Format [1], the ISO equivalent [4], and other data formats, including finger images.

The `DataInterchange::ANSI2004Record` represents all the finger views contained in a pair of

ANSI 2004 fingerprint([1]) and finger image ([2]) records. This class supports the insert/update/remove of finger views from the data interchange record, enabling the runtime updating of the object. In addition, the encoded format of the minutia record can be obtained, enabling the read/modification/write of the record.

*(**NOTE:** Reading data from finger image records is not currently supported)*

Listing 19.2: ANSI 2004 Data Interchange

```
#include <iostream>
#include <be_data_interchange_ansi2004.h>

using namespace std;
using namespace BiometricEvaluation;
using namespace BiometricEvaluation::Framework::Enumeration;

void
printViewInfo(Finger::INCITSView &fngv)
{
        cout << "Begin ----------------------------------------" << endl;
        cout << "Image resolution is " << fngv.getImageResolution() << endl;
        cout << "Image size is " << fngv.getImageSize() << endl;
        cout << "Image depth is " << fngv.getImageColorDepth() << endl;
        cout << "Compression is " << fngv.getCompressionAlgorithm() << endl;
        cout << "Scan resolution is " << fngv.getScanResolution() << endl;

        cout << "Finger position is " << fngv.getPosition() << endl;
        cout << "Impression type is " << fngv.getImpressionType() << endl;
        cout << "Quality is " << fngv.getQuality() << endl;
        cout << "Eqpt ID is " << hex << showbase << fngv.getCaptureEquipmentID() << endl;
        cout << dec;

        Feature::INCITSMinutiae fmd = fngv.getMinutiaeData();
        cout << "Minutiae format is " << to_string(fmd.getFormat()) << endl;
        cout << "There are " << fmd.getMinutiaPoints().size()
             << " minutiae points." << endl;
        cout << "End ------------------------------------" << endl;
}

bool
showAllViews(const DataInterchange::ANSI2004Record &record)
{
        if (record.getNumFingerViews() == 0) {
                cout << "No finger views present.\n";
                return (true);
        }
        for (int i = 1; i <= record.getNumFingerViews(); i++) {
                cout << "++++++++++++++++++++++++++++++\n";
                cout << "View number " << i << ":\n";
                auto fngv = record.getView(i);
                printViewInfo(fngv);
                cout << "Test getMinutia(): View " << i << " has "
                    << record.getMinutia(i).getMinutiaPoints().size()
                    << " minutiae points.\n";
        }
        return (true);
}

```

```
50 int
51 main(int argc, char* argv[])
52 {
53         std::unique_ptr<DataInterchange::ANSI2004Record> record;
54
55         /* Construct with a file, minutia record only. */
56         try {
57                 record.reset(new DataInterchange::ANSI2004Record(
58                     "test_data/fmr.ansi2004", ""));
59         } catch (Error::Exception& e) {
60                 cout << "A file error occurred: " << e.what() << endl;
61                 return (EXIT_FAILURE);
62         }
63
64         /* Remove all views but the first */
65         record->isolateView(1);
66         showAllViews(*record);
67
68         /* Modify the minutia in a finger view */
69         auto minutiaRecord = record->getMinutia(1);
70         auto minutiaPoints = minutiaRecord.getMinutiaPoints();
71         for (auto& fm: minutiaPoints) {
72                 fm.coordinate.x += 10;
73                 fm.coordinate.y += 10;
74         }
75         /* Replace minutiae in the remaining view */
76         minutiaRecord.setMinutiaPoints(minutiaPoints);
77         record->setMinutia(1, minutiaRecord);
78         showAllViews(*record);
79
80         /* Obtain the ANSI-378 record and instantiate an object from it */
81         auto fmr = record->getFMR();
82         BE::Finger::ANSI2004View fmrView(fmr, Memory::uint8Array{}, 1);
83         /* The fmr object can also be written to a file */
84
85         return (EXIT_SUCCESS);
86 }
```

Chapter 20

Messaging

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

20.1 Message Center

`Process::MessageCenter` is the public-facing class an application uses to receive messages over a network. A *message* is a user-defined blob of data stored in an array of bytes. Instantiate a `MessageCenter`, and it will dilligently await connections on the specified port in a separate process. During its run-loop, the apppplication may poll or wait to determine if a message is waiting. The application has the choice of dealing with the message, sending a response, or ignoring the message entirely. Because the `MessageCenterListener` is in a separate process, the main run-loop of the application does not have to be interrupted. The `MessageCenter` classes utilize existing framework inter-process communication techniques to propagate messages (see Subsection 9.2.4 on page 33).

Listing 20.1: Basic `MessageCenter` Usage

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

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

20.2 Command Center

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

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

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

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

Listing 20.2: Basic `CommandCenter` Usage

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

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

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

Chapter 21

Parallel Processing

21.1 MPI Parallel Processing Package

The `MPI` package is a set of APIs used implement parallel processing using the MPI [21] network-based messaging system. The core concept implemented in the framework is that of a distributor, one or more receivers, work packages, and a processing element to be implemented by the application.

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

Figure 21.1 on the following page shows the processes and data flow for a typical parallel job using components of the Framework. The distributor process (Task-0) executes code from the `Distributor` class, and the receiver processes (Task-N) execute `Receiver` class code. Within each process is shown the Framework packages that could be used for the job. The *Lib* element refers to the "black-box" component of software being tested, a fingerprint matching library, for example. In this example, a record store is used as the data source, and record keys are sent in the work packages. On the receiving side, the keys are used to read record data (values) from the same store.

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

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

Second, each Task-N can perform some work before creating the workers. One example is the loading of a large data set into memory; again, this is done within the application. Because Task-N calls the POSIX function `fork()` to create the workers, each worker inherits the work done by Task-N. In the case of a memory load, each worker now has that memory mapped into it's address space. See Section 21.7 on page 77 for more details.

21.2 Work Package

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

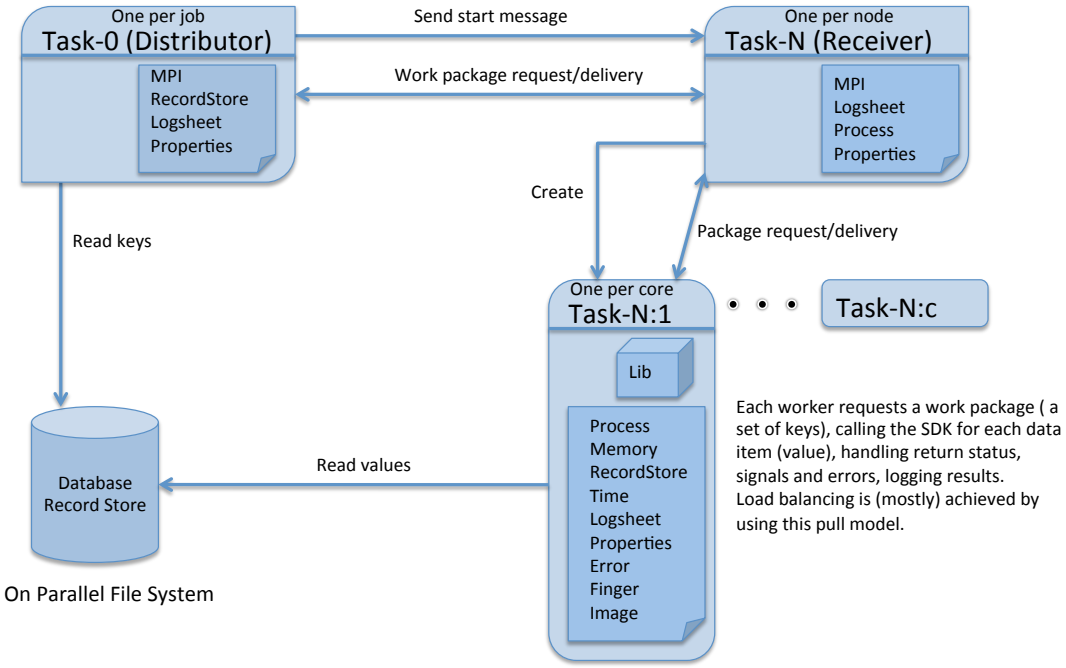


Figure 21.1: MPI Parallel Job Processes and Data Flow

The classes `RecordStoreDistributor` (Section 21.5.1 on the following page) and `RecordProcessor` (Section 21.7.1 on page 78) are examples of `WorkPackage` clients that insert and remove data from a work package.

21.3 MPI Resources

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

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

Workers Per Node Used by the receiver process to start the required number of workers. This value is either an integer string, or one of the special values:

NUMCPUS The number of logical CPUs, also known as hyperthreads;

NUMCORES The number of processing cores;

NUMSOCKETS The number of physical processor sockets.

Choosing the number of workers based on one of the special values depends on whether the processing is to take advantage of instruction pipelines, cache hierarchies, or other features of the processor hardware implementation.

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

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

Checkpoint Path Used by the distributor process to place checkpoint files. This property is required when checkpointing is enabled, otherwise ignored.

Subclasses and other components of the MPI Framework may add properties as needed, usually to the same file as the above properties.

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

21.4 Checkpoint Save and Restore

The MPI package supports checkpointing, where the state of an MPI job can be saved and restored. In the current implementation, checkpoints simply save information about the last work package that was distributed. Hence, the `Distributor` classes (See Section 21.5 on the next page) are responsible for saving this state. As a consequence, a distributor cannot record whether the work package was actually processed. When checkpointing is enabled, the resources (see Section 21.3) file for the job must contain the `Checkpoint Path` property.

A checkpoint is saved when the application enables the checkpoint capability via the `Runtime` (21.8) object, and a clean shutdown is performed by sending a signal to the distributor process, Task-0.

A checkpoint is restored when the application enables checkpointing, and the checkpoint file is opened. If restore is requested, but no checkpoint file is present, the job starts in the same manner as if checkpointing was not enabled.

While the MPI job is running, the checkpoint file will minimally contain the process ID of the Task-0 distributor. A job script or other program can use this ID to shutdown the job with a complete checkpoint. An example command is:

```
kill -QUIT `cat /tmp/Distributor.chk | grep PID | cut -d= -f2`
```

21.5 Distributor

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

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

For distributors, a basic checkpoint file is created. This text file is a set of key-value pairs. The `Distributor` class writes this information to the checkpoint file:

PID The process ID of the Task-0 distributor. This value is written on job startup and can be used to send the clean shutdown signal to the process.

21.5.1 Record Store Distributor

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

An application can create an instance of a `RecordStoreDistributor` with the name of a record store in order to distribute records for processing across the MPI job. Listing 21.3 on page 83 shows an example section of code to create a record store distributor. In this type of application there is no need for the application code to refine any of the Framework classes.

Class `RecordStoreDistributor` has these additional MPI resources:

Input Record Store The input record store,

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

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

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

The `RecordStoreDistributor` class writes this information to the checkpoint file:

Reason A string describing the reason the checkpoint was taken.

Last Key The last record store key that was distributed.

Num Keys The number of keys that were distributed.

21.5.2 CSV Distributor

`CSVDistributor` reads text lines from an input file with no other semantic reasoning. The lines of the file are distributed in the work package containing an application-defined line count. Additional features of the `CSVDistributor` class include randomizing the input, reading the entire file into a buffer before distribution begins, and checkpoint support.

Class `CSVDistributor` has these additional MPI resources:

Input CSV The input CSV file.

Chunk Size How many lines of the file to distribute in a work package.

Read Entire File Read the entire file into buffer; "YES" or "NO".

CSV Delimiter Character delimiter used to tokenize lines of the file.

Randomize Lines Whether to randomize distribution of the data;"YES" or "NO"

Random Seed Integer value used to seed the random function.

Trim CSV Whitespace Whether to trim white space from the input lines; "YES" or "NO".

The `CSVDistributor` class writes this information to the checkpoint file:

Reason A string describing the reason the checkpoint was taken.

Line Count The number of lines from the CSV file that were distributed.

Random Seed The seed that was used to randomize the CSV file. Required when the "Random Seed" property is used in the distributor resources file.

21.6 Receiver

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

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

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

21.7 Work Package Processor

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

Implementations of this class can be considered to have dual responsibilities. First is the management of common state used by all workers (Task-N:c in Figure 21.1 on page 74); creating state data shared by all workers, for example. Second, as a factory to create a package consumer for the worker process.

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

`newProcessor()` returns a new instance of the package processor. This method is called by the Framework when a new process is started by the receiver to consume work packages sent by the distributor. This method is a factory, creating new instances of the `WorkPackageProcessor` implementation. Therefore, it must create a "fully-formed" object that may have different state than that created by the class constructor. An example would be creating an output log file with record information. This output file would not be created in the constructor because the object returned from that will not process a work package; it is the factory object.

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

The `performShutdown()` method is optionally implemented by the application to take action after all the work packages have been distributed, and is called by the framework after all the workers have terminated. The default implementation of this method does nothing.

21.7.1 Record Processor

`RecordProcessor` is a partial implementation of `WorkPackageProcessor` and defines the `processWorkPackage()` of the `WorkPackageProcessor` interface; other methods are declared as pure-virtual and must be implemented by a child class. In addition, `RecordProcessor` declares a new pure-virtual method, `processRecord()` to be implemented by a subclass to process a single record from the record store. In summary, `RecordProcessor` removes records from the work package to be processed within the subclass, which is defined by the application. See Listing 21.1 on the next page and Listing 21.2 on page 80 for a example of such an implementation.

21.8 MPI Runtime

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

One optional parameter to the `Runtime` constructor control the checkpoint capability (see 21.4 and Listing 21.2 on page 80):

checkpointEnable Write a checkpoint file when a clean shutdown is requested and restore from a checkpoint if the file is present. This parameter defaults to `false`.

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

SIGQUIT Exit when the current work package is exhausted (“clean exit”);

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

SIGTERM Exit immediately (“termination exit”).

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

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

In addition to sending signals from outside the process, a worker can shutdown itself or the entire job through exceptions. Any type of exception thrown from within a worker will cause that individual worker to shutdown, and its status will be communicated up the chain. A special type of exception, `TerminateJob`, will shutdown the individual worker, and additionally communicate up the chain to the distributor that all other workers should immediately exit. Throwing `TerminateJob` from a worker is similar in result to sending `SIGTERM` to a distributor.

21.9 Logging

In order to aid tracing and debugging of a parallel job, the MPI Framework can be configured to write trace messages to the log storage. These trace messages are logged as debug messages instead of normal entries. The type and location of the log is given to the Framework by using a URL as a property when starting the MPI job (see Section 21.3 on page 75).

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

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

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

21.10 MPI Framework Applications

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

Listing 21.1 shows the header file that declares a subclass of `RecordProcessor`. The `newProcessor()`, `performInitialization()`, and `processRecord()` methods are those required to complete an implementation of `RecordProcessor`. A memory buffer pointer is managed with a smart pointer object.

Listing 21.1: MPI Framework Application Classes

```
1  class TestRecordProcessor : public BiometricEvaluation::MPI::RecordProcessor {
2  public:
3          /**
4           * @brief
5           * The property string ``Logsheet URL''.
6           */
7          static const std::string RECORDLOGSHEETURLPROPERTY;
8
9          static const uint32_t SHAREDMEMORYSIZE = 2048;
10
11         TestRecordProcessor(
12             const std::string &propertiesFileName);
13         ~TestRecordProcessor();
14
15         std::shared_ptr<BE::MPI::WorkPackageProcessor>
16         newProcessor(std::shared_ptr<BE::IO::Logsheet> &logsheet);
17
18         void
19         performInitialization(std::shared_ptr<BE::IO::Logsheet> &logsheet);
20
21         void processRecord(const std::string &key);
22
23         void processRecord(
24             const std::string &key,
25             const BE::Memory::uint8Array &value);
```

```
26
27          void
28          performShutdown();
29  protected:
30  private:
31          std::shared_ptr<BE::IO::Logsheet> _recordLogsheet;
32          std::shared_ptr<char> _sharedMemory;
33          uint32_t _sharedMemorySize;
34  };
```

Next, Listing 21.2 shows the implementation of the class methods. In this simple example, each record is acknowledged with a log entry.

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

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

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

Listing 21.2: MPI Framework Application Implementation

```
1  #include <be_mpi_receiver.h>
2  #include <be_mpi_recordstoredistributor.h>
3  #include <be_mpi_runtime.h>
4
5  #include "test_be_mpi.h"
6
7  using namespace BiometricEvaluation;
8
9  static const std::string DefaultPropertiesFileName("test_be_mpi.props");
10
11 /*
12  * Implementations of the MPI RecordProcessor class interface.
13  * Calls the parent constructor to manage the properties file name.
14  */
15 TestRecordProcessor::TestRecordProcessor(
16     const std::string &propertiesFileName) :
17     RecordProcessor(propertiesFileName)
18 {
19 }
20
21 TestRecordProcessor::~TestRecordProcessor()
22 {
23 }
24
25 /*
26  * Factory object: Log our call and set up the shared memory buffer.
27  */
28 void
29 TestRecordProcessor::performInitialization(
30     std::shared_ptr<IO::Logsheet> &logsheet)
31 {
```

```
32          this->setLogsheet(logsheet);
33
34          /*
35           * Set up the memory that will be shared across all instances.
36           */
37          char *buf = (char *)malloc(SHAREDMEMORYSIZE);
38          strcpy(buf, "SHARED MEMORY");
39          this->_sharedMemorySize = SHAREDMEMORYSIZE;
40          this->_sharedMemory = std::unique_ptr<char>(buf);
41
42          *logsheet.get() << std::string(__FUNCTION__) << " called: ";
43          *logsheet.get()
44              << "Shared memory size is " << this->_sharedMemorySize
45              << " and contents is [" << buf << "]";
46          BE::MPI::logEntry(*logsheet.get());
47  }
48
49  /*
50   * Factory object: Create a new instance of the TestRecordProcess
51   * that will work on work package records. Each instance gets
52   * its own instance of the log sheet.
53   */
54  std::shared_ptr<BiometricEvaluation::MPI::WorkPackageProcessor>
55  TestRecordProcessor::newProcessor(
56      std::shared_ptr<IO::Logsheet> &logsheet)
57  {
58          std::string propertiesFileName =
59              this->getResources()->getPropertiesFileName();
60          TestRecordProcessor *processor =
61              new TestRecordProcessor(propertiesFileName);
62          processor->setLogsheet(logsheet);
63
64          /*
65           * If we have our own Logsheet property, and we can open
66           * that Logsheet, use it for record logging; otherwise,
67           * create a Null Logsheet for these events. We use the
68           * framework's Logsheet for tracing of processing, not
69           * record handling logs.
70           */
71          std::string url;
72          std::unique_ptr<BE::IO::PropertiesFile> props;
73          try {
74                  /* It is crucial that the Properties file be
75                   * opened read-only, else it will be rewritten
76                   * when the unique ptr is destroyed, causing
77                   * a race condition with other processes that
78                   * are reading the file.
79                   */
80                  props.reset(new BE::IO::PropertiesFile(
81                      propertiesFileName, IO::READONLY));
82                  url = props->getProperty(
83                      TestRecordProcessor::RECORDLOGSHEETURLPROPERTY);
84          } catch (BE::Error::Exception &e) {
85                  url = "";
86          }
87          processor->_recordLogsheet = BE::MPI::openLogsheet(
```

```
88              url, "Test Record Processing");
89          processor->_sharedMemory = this->_sharedMemory;
90          processor->_sharedMemorySize = this->_sharedMemorySize;
91
92          std::shared_ptr<BiometricEvaluation::MPI::WorkPackageProcessor> sptr;
93          sptr.reset(processor);
94          return (sptr);
95  }
96
97  /*
98   * Helper function to log some information about a record.
99   */
100 static void
101 dumpRecord(
102     BE::IO::Logsheet &log,
103     const std::string key,
104     const Memory::uint8Array &val)
105 {
106         log << "Key [" << key << "]: ";
107         /* Dump some bytes from the record */
108         for (uint64_t i = 0; i < 8; i++) {
109                 log << std::hex << (int)val[i] << " ";
110         }
111         log << " |";
112         for (uint64_t i = 0; i < 8; i++) {
113                 log << (char)val[i];
114         }
115         log << "|";
116         BE::MPI::logEntry(log);
117 }
118
119 /*
120  * The worker object: Log to the Framework Logsheet, obtain the data for
121  * the record, and log some information to the record Logsheet.
122  */
123 void
124 TestRecordProcessor::processRecord(const std::string &key)
125 {
126         BE::IO::Logsheet *log = this->getLogsheet().get();
127
128         if (this->getResources()->haveRecordStore() == false) {
129                 BE::MPI::logMessage(*log, "processRecord(" + key + ")"
130                     + " called but have no record store; returning.");
131                 return;
132         }
133         *log << "processRecord(" << key << ") called: ";
134         char *buf = this->_sharedMemory.get();
135         *log << "Shared memory size is " << this->_sharedMemorySize
136             << " and contents is [" << buf << "]";
137         BE::MPI::logEntry(*log);
138
139         Memory::uint8Array value(0);
140         std::shared_ptr<IO::RecordStore> inputRS =
141             this->getResources()->getRecordStore();
142         try {
143                 inputRS->read(key, value);
```

```
144         } catch (Error::Exception &e) {
145                 *log << string(__FUNCTION__) <<
146                     " could not read record: " <<
147                     e.whatString();
148                 return;
149         }
150         /*
151          * Log record info to our own Logsheet instead of
152          * the one provided by the framework.
153          */
154         BE::IO::Logsheet *rlog = this->_recordLogsheet.get();
155         dumpRecord(*rlog, key, value);
156 }
157
158 /*
159  * The worker object: Log to the Framework Logsheet, and log some record
160  * information to the record Logsheet.
161  */
162 void
163 TestRecordProcessor::processRecord(
164     const std::string &key,
165     const BiometricEvaluation::Memory::uint8Array &value)
166 {
167         BE::IO::Logsheet *log = this->getLogsheet().get();
168         *log << "processRecord(" << key << ", [value]) called: ";
169         char *buf = this->_sharedMemory.get();
170         *log << "Shared memory size is " << this->_sharedMemorySize
171             << " and contents is [" << buf << "]";
172         BE::MPI::logEntry(*log);
173
174         /*
175          * Log record info to our own Logsheet instead of
176          * the one provided by the framework.
177          */
178         BE::IO::Logsheet *rlog = this->_recordLogsheet.get();
179         dumpRecord(*rlog, key, value);
180 }
181
182 /*
183  * Factory object: Log our call.
184  */
185 void
186 TestRecordProcessor::performShutdown()
187 {
188         std::shared_ptr<BE::IO::Logsheet> logsheet = this->getLogsheet();
189         *logsheet.get() << std::string(__FUNCTION__)
190             << " called in PID " << getpid() << ": ";
191         BE::MPI::logEntry(*logsheet.get());
192 }
```

Listing 21.3: MPI Framework Application Main

```
1 int
2 main(int argc, char* argv[])
3 {
4         /*
```

```
5           * Process optional checkpoint and include-values flags.
6           */
7          bool cpEnable{false}, includeValues{false};
8          char ch;
9          while ((ch = getopt(argc, argv, "cv")) != -1) {
10                 switch (ch) {
11                         case 'r': cpEnable = true; break;
12                         case 'v': includeValues = true; break;
13                 }
14         }
15         MPI::Runtime runtime(argc, argv, cpEnable);
16         std::unique_ptr<MPI::RecordStoreDistributor> distributor;
17         std::unique_ptr<MPI::Receiver> receiver;
18         std::shared_ptr<TestRecordProcessor> processor;
19
20         if (includeValues) {
21                 MPI::printStatus("Test Distributor and Receiver, keys and values");
22         } else {
23                 MPI::printStatus("Test Distributor and Receiver, keys only");
24         }
25         try {
26                 distributor.reset(
27                     new MPI::RecordStoreDistributor(propFile, includeValues));
28                 processor.reset(new TestRecordProcessor(propFile));
29                 receiver.reset(new MPI::Receiver(propFile, processor));
30                 runtime.start(*distributor, *receiver);
31                 runtime.shutdown();
32         } catch (Error::Exception &e) {
33                 MPI::printStatus("Caught: " + e.whatString());
34                 runtime.abort(EXIT_FAILURE);
35         } catch (...) {
36                 MPI::printStatus("Caught some other exception");
37                 runtime.abort(EXIT_FAILURE);
38         }
39
40         return (EXIT_SUCCESS);
41 }
```

References

[1] *ANSI INCITS 378-2004: Finger Minutiae Format for Data Interchange*. ANSI/INCITS, 2004. 49, 55, 65, 66

[2] *ANSI INCITS 381-2004: Finger Image-Based Data Interchange Format*. ANSI/INCITS, 2004. 66

[3] *ANSI INCITS 385-2004: Face Recognition Format for Data Interchange*. ANSI/INCITS, 2004. 59

[4] *ISO/IEC 19794-2: Information technology - Biometric data interchange formats - Part 2: Finger minutiae data*. ISO/IEC, first edition, 2005. 49, 65

[5] *ISO/IEC 7816-4: Identification cards - Integrated circuit cards - Part 4: Organization, security and commands for interchange*. ISO/IEC, second edition, 2005. 45

[6] *American National Standard for Information Systems - Data Format for the Interchange of Fingerprint, Facial, & Other Biometric Information*. ANSI/NIST-ITL, 1-2007 edition, 2007. NIST Special Publication 500-271. 4, 47, 52, 53, 63, 88

[7] *American National Standard for Information Systems - Data Format for the Interchange of Fingerprint, Facial, & Other Biometric Information*. ANSI/NIST-ITL, 1-2011 edition, 2011. NIST Special Publication 500-290. 47, 57

[8] Mark Adler. zlib, 2012. http://www.zlib.net. 20

[9] Berkeley DB. https://sqlite.org. 89

[10] CMake. https://cmake.org. 87

[11] World Wide Web Consortium. Portable Network Graphics Standard, 2003. http://www.w3.org/TR/PNG/. 39

[12] FFmpeg Multimedia Framework. http://www.ffmpeg.org. 41, 88

[13] GNU Make Project. http://www.gnu.org/software/make. 87

[14] Independent JPEG Group. libjpeg, 2011. http://www.ijg.org/. 38, 88

[15] Joint Photographic Experts Group. JPEG2000 Image Standard, 1992. http://www.jpeg.org/jpeg2000/index.html. 38

[16] Joint Photographic Experts Group. JPEG Image Standard, 2011. http://www.jpeg.org/jpeg/index.html. 38

[17] International Committee for Information Technology Standards. http://www.incits.org. 55, 65

[18] ISO/IEC Joint Technical Committee 1/SC 37 Biometrics. 55

[19] Communications and Remote Sensing Lab, Université catholique de Louvain. OpenJPEG Library, 2011. http://www.openjpeg.org/. 38, 88

[20] Microsoft. Bitmap Image File Format. https://en.wikipedia.org/wiki/BMP_file_format. 39

[21] Message Passing Interface Forum. http://www.mpi-forum.org. 4, 73, 88

[22] NIST Biometric Image Software, 2011. https://www.nist.gov/services-resources/software/nist-biometric-image-software-nbis. 7, 38, 39, 88

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

[24] The Open MPI Project. http://www.openmpi.org. 89, 91

[25] The OpenSSL Project. http://www.openssl.org. 88

[26] Greg Roelofs. libpng, 2011. http://www.libpng.org/pub/png/libpng.html. 39, 88

[27] Sam Leffler. LibTIFF – TIFF Library and Utilities, May 2017. http://www.simplesystems.org/libtiff/. 39, 88

[28] The SQLite Project. https://sqlite.org. 89

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

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

[31] Wavelet Scalar Quantization Gray-Scale Fingerprint Image Compression Standard, 2010. https://www.fbibiospecs.cjis.gov/Document/Get?fileName=WSQ_Gray-scale_Specification_Version_3_1_Final.pdf. 39, 88

Appendix A

Building the Framework

A.1 Language Features

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

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

- GNU Compiler Collection version 4.8.5 on Linux.
- Apple LLVM version 11.0.3 (clang-1103.0.32.59) on MacOS.

A.2 The Framework Build System

The distribution of BECommon includes a set of `make` files used to build the BECommon library, as well as install the library and header files. These `make` file use some features of the GNU make [13] system, and therefore the GNU software must be installed on the user's system. Future versions of BECommon may use a different build system.

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

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

A.3 The CMake Build System

Building the BECommon using CMake [10] is possible, and provides a simpler cross-platform build system. In the `common/src/libbiomeval` directory is a `CMakeLists.txt` file that controls the build. Advantages of using the CMake build system include auto-discovery of required software dependencies and compilation of the test programs. Also, some dependencies are optional, and when not found, the library will be built without some functionality. For example, video support and MPI parallel job support are optional under the CMake build.

To build static and shared library versions of libbiomeval, including the subset of NBIS included with the Framework, the steps are:

1. Create a build directory; in this example, it will be under libbiomeval:

```
mkdir build; cd build
```

2. Run CMake using the `CMakeLists.txt`:

```
cmake ..
```

3. Build the Framework:

```
make
```

4. Install static and shared libraries plus headers:

```
make install
```

5. Create an RPM on CentOS or RedHat Linux systems:

```
make package
```

To build the debug version of the library, substitute for step 2:

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

To use a different compiler for the MPI component (Intel, OpenMPI are among the supported compilers), substitute for step 2:

```
cmake -DMPI_CXX_COMPILER=mpiicpc ..
```

A.4 External Software Dependencies

The Biometric Evaluation Framework is built upon several other software packages. The packages are used for image processing, biometric data record formats, the message passing interface [21], as well as operating system and compiler tool chains.

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

A.4.1 NIST Biometric Image Software

The NIST Biometric Image Software (NBIS) [22] is a set of packages used for ANSI-NIST [6], WSQ [31] formats, and other support. The BECommon uses NBIS to process these biometric record formats. and contains a subset of the NBIS packages. Therefore there is no need to install NBIS. However, the BECommon build system supports using an installed NBIS package as an alternative.

A.4.2 Video and Image Processing

For the `Image` classes, the JPEG [14], NBIS [22], OpenJPEG [19], PNG [26], and TIFF [27] development libraries are required.

For `Video` classes, the FFmpeg [12] libraries are used. When building from source, configure to build and install shared libraries. By default, only static libraries are built.

A.4.3 Cryptography

Cryptography support is provided by the OpenSSL [25] library. An example is the `openssl-devel` package on Linux systems which provides the `libcrypto` file and associated header files for development.

A.4.4 Sqlite

SQLite is an embedded Structured Query Language (SQL) database engine and is used by the `IO::SQLiteRecordStore` class to provide an `IO::RecordStore` that is backed by a SQLite database. Information on SQLite can be found at [28].

A.4.5 Berkeley Database

The Berkeley Database BDB [9] is available as both open source and closed source commercial variants. The BECommon class `IO::DBRecordStore` uses the BDB software to store key/value pairs. There are two versions of the BDB API; BECommon uses version 1.85 as defined in the original open source distribution.

A.4.6 Message Passing Interface

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

One common implementation of MPI is OpenMPI [24], available as source code, or binary packages. Often the MPI runtime is a separate binary package from the MPI development software. As an example, for many Linux distributions, an example of the runtime package is `openmpi-1.6.4-3`, while the related development package would be `openmpi-devel-1.6.4-3`.

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

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

Appendix B

Running an MPI Job

B.1 OpenMPI

This chapter describes how to use the OpenMPI [24] runtime system to execute an MPI job. Some parameters passed to the `mpirun` command are related to the notions captured in the Biometric Evaluation Framework MPI support.

B.2 Example Shell Script

Listing B.1: Example Script to run MPI

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

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

```
29 EOF
30
31 #
32 # Two forms of the nodes string, one for the script to copy all
33 # files out, one for the mpirun command.
34 #
35 NODES="node01b node02b node03b node04b"
36 MPINODES="node01b,node02b,node03b,node04b"
37
38 #
39 # MPIPROCS must be >= 2, is the Task-N count plus one for Task-0.
40 #
41 MPIPROCS=5
42
43 #
44 # Set any options to the OpenMPI mpirun command. The example below will
45 # turn on some tracing and how processes are mapped to nodes.
46 #
47 #MPIOPTS=" --show-progress --debug-daemons --display-devel-map"
48
49 # Where the program is run. The directory must exist on all the
50 # nodes, and this script must be started here.
51 DIR=$PWD
52
53 #
54 # LIBS is any libraries th must coexist with the program to be run.
55 #
56 LIBS=
57 PROGRAM=test_mpi
58 CPFILES="$PROGRAM $PROPS $LIBS"
59
60 #
61 # The test program and dependencies must exist on all nodes, so copy
62 # everything to the runtime directory on all nodes. It helps to run
63 # an SSH agent or something similar.
64 #
65 for n in $NODES; do
66         echo $n;
67         scp -p $CPFILES $n:$DIR;
68 done
69
70 #
71 # Run the program as an MPI job. mpirun must be in the users path.
72 # The properties file name is the only parameter to the program.
73 #
74 EXECSTR="$PROGRAM $PROPS"
75 mpirun $MPIOPTS -H $MPINODES -np $MPIPROCS --path $DIR $EXECSTR
```

Appendix C

Namespace Index

C.1 Namespace List

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

Appendix D

Hierarchical Index

D.1 Class Hierarchy

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

Appendix E

Class Index

E.1 Class List

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

Appendix F

File Index

F.1 File List

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

Appendix G

Namespace Documentation

G.1 BiometricEvaluation Namespace Reference

Namespaces

- namespace **Error**

  Exceptions, and other error handling.
- namespace **Face**

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

  Biometric information relating to biometric features not specific to any type of biometric record.
- namespace **Finger**

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

  Information about the framework.
- namespace **Image**

  Basic information relating to images.
- namespace **IO**

  Input/Output functionality.
- namespace **Iris**

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

  Support for memory-related operations.
- namespace **MPI**

  Common declarations and functions for the MPI-based functionality.
- namespace **Palm**

  Biometric information relating to palm images and derived information.
- namespace **Plantar**

  Biometric information relating to plantar images and derived information.
- namespace **Process**

  Process (p. 170) *information and controls.*
- namespace **System**

  Operating system, hardware, etc.
- namespace **Text**

Text *(p. 173) processing for string objects.*

- namespace **Time**

  Support for time and timers.

- namespace **Video**

  Basic information relating to video and streams.

- namespace **View**

  View *(p. 819) information.*

G.1.1 Detailed Description

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

This software was developed at the National Institute of Standards and Technology (NIST) by employees of the Federal Government in the course of their official duties. Pursuant to title 17 Section 105 of the United States Code, this software is not subject to copyright protection and is in the public domain. NIST assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. An interface to the object that processes a package of work from the **MPI** (p. 162) Receiver.

G.2 BiometricEvaluation::Error Namespace Reference

Exceptions, and other error handling.

Classes

- class **ConversionError**

  Error *(p. 112) when converting one object into another, a property value from string to int, for example.*

- class **DataError**

  Error *(p. 112) when reading data from an external source.*

- class **Exception**

  The parent class of all ***BiometricEvaluation*** *(p. 111) exceptions.*

- class **FileError**

  File error when opening, reading, writing, etc.

- class **MemoryError**

  An error occurred when allocating an object.

- class **NotImplemented**

  A ***NotImplemented*** *(p. 636) object is thrown when the underlying implementation of this interface has not or could not be created.*

- class **ObjectDoesNotExist**

  The named object does not exist.

- class **ObjectExists**

  The named object exists and will not be replaced.

- class **ObjectIsClosed**

  The object is closed.

- class **ObjectIsOpen**

The object is already opened.

- class **ParameterError**

  An invalid parameter was passed to a constructor or method.

- class **SignalManager**

  *A **SignalManager** (p. 759) object is used to handle signals that come from the operating system.*

- class **StrategyError**

  *A **StrategyError** (p. 789) object is thrown when the underlying implementation of this interface encounters an error.*

Functions

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

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

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

G.2.1 Detailed Description

Exceptions, and other error handling.

The **Error** (p. 112) package contains classes for exceptions, and functions used for error handling, including signals generated by a process.

G.2.2 Function Documentation

G.2.2.1 errorStr()

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

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

Parameters

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

Returns

The current error message specified by errno.

G.3 BiometricEvaluation::Face Namespace Reference

Biometric information relating to face images and derived information.

Classes

- class **INCITSView**

  A class to represent single facial image view and derived information.
- class **ISO2005View**

  A class to represent single face view and derived information.
- struct **PoseAngle**

  Representation of pose angle and uncertainty.

Typedefs

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

Enumerations

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

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

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

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

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

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

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

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

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

Source type codes.

G.3.1 Detailed Description

Biometric information relating to face images and derived information.

The **Face** (p. 113) package gathers all face related matters, including classes to represent face information and helper functions for conversion between biometric representations. Contained within this namespace are classes to represent specific record formats, such as ISO 19794-5.

G.3.2 Typedef Documentation

G.3.2.1 PropertySet

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

A set of properties.

G.4 BiometricEvaluation::Feature Namespace Reference

Biometric information relating to biometric features not specific to any type of biometric record.

Namespaces

- namespace **Sort**

Classes

- class **AN2K7Minutiae**

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

  Representation of the core.
- struct **DeltaPoint**

  Representation of the delta.
- struct **FrictionRidgeGeneralizedPosition**

  Representation of the position (Finger/Palm/Plantar) used in this class and child classes.
- class **INCITSMinutiae**

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

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

  Representation of a finger minutiae data point.
- struct **MPEGFacePoint**

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

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

Typedefs

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

Enumerations

- enum class **PositionType** { **Finger** = 0 , **Palm** = 1 , **Plantar** = 2 }

  Enumeration of the types of position classes used in this class and child classes.
- enum class **MinutiaeFormat** {
  AN2K7 = 0 , **IAFIS** , **Cogent** , **Motorola** ,
  Sagem , **NEC** , **Identix** , **M1** ,
  Other }

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

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

  Enumerate the types of extraction methods for ridge counts.

Functions

- std::ostream & **operator**<< (std::ostream &s, const **Feature::FGP** &fgp)

  *Output stream overload for **FrictionRidgeGeneralizedPosition** (p. 464).*
- std::ostream & **operator**<< (std::ostream &, const **AN2K7Minutiae::FingerprintReadingSystem** &)

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

G.4.1 Detailed Description

Biometric information relating to biometric features not specific to any type of biometric record.

Definition of an MPEG4 **Face** (p. 113) feature point. See ISO/IEC 14496-2.

G.4.2 Typedef Documentation

G.4.2.1 AN2K7MinutiaeSet

```
using BiometricEvaluation::Feature::AN2K7MinutiaeSet
```

Initial value:

```
std::vector<std::shared_ptr<AN2K7Minutiae>>
```

G.4.3 Function Documentation

G.4.3.1 operator<<()

```
std::ostream & BiometricEvaluation::Feature::operator<< (
            std::ostream & s,
            const  Feature::FGP & fgp)
```

Output stream overload for **FrictionRidgeGeneralizedPosition** (p. 464).

Parameters

in	*s*	Stream on which to append formatted information.
in	*fgp*	**FrictionRidgeGeneralizedPosition** (p. 464) information to append to stream.

Returns

stream with a fgp textual representation appended.

G.5 BiometricEvaluation::Feature::Sort Namespace Reference

Classes

- class **Angle**
- class **Polar**

Sort *(p. 117) by increasing distance from center and angle (theta).*

- class **Quality**
- class **XY**
- class **YX**

Enumerations

- enum class **Kind** {
  XYAscending , **XYDescending** , **YXAscending** , **YXDescending** ,
  QualityAscending , **QualityDescending** , **AngleAscending** , **AngleDescending** ,
  PolarCOMAscending , **PolarCOMDescending** , **PolarCOIAscending** , **PolarCOIDescending** ,
  Unknown }

Functions

- void **updateIndicies** (BiometricEvaluation::Feature::MinutiaPointSet &mps)

  Renumber index numbers in a MinutiaPointSet in place.
- std::vector< **Feature::MinutiaPoint** > **sort** (std::vector< **Feature::MinutiaPoint** > &minutia, const **Kind** &sortOrder)

  Sort *(p. 117) minutia.*
- std::vector< **Feature::MinutiaPoint** > **stableSort** (std::vector< **Feature::MinutiaPoint** > &minutia, const **Kind** &sortOrder)

  Sort *(p. 117) minutia, maintaining existing order if elements are otherwise deemed equal.*

G.5.1 Detailed Description

Utilities for sorting MinutiaPointSets.

G.5.2 Enumeration Type Documentation

G.5.2.1 Kind

`enum class` **`BiometricEvaluation::Feature::Sort::Kind`** `[strong]`

Sort (p. 117) order of MinutiaPointSets.

Enumerator

XYAscending	Lowest to highest X value, followed by Y value.
XYDescending	Highest to lowest X value, followed by Y value.

Enumerator

YXAscending	Lowest to highest Y value, followed by X value.
YXDescending	Highest to lowest Y value, followed by X value.
QualityAscending	Lowest to highest quality value.
QualityDescending	Highest to lowest quality value.
AngleAscending	Lowest to highest angle (theta) value.
AngleDescending	Highest to lowest angle (theta) value.

Enumerator

PolarCOMAscending	Lowest to highest distance from center of minutia mass, followed by angle (theta).
PolarCOMDescending	Highest to lowest distance from center of minutia mass, followed by angle (theta).
PolarCOIAscending	Lowest to highest distance from center of image, followed by angle (theta).

Enumerator

PolarCOIDescending	Highest to lowest distance from center of img, followed by angle (theta).
Unknown	**Sort** (p. 117) order cannot be determined.

G.5.3 Function Documentation

G.5.3.1 sort()

```
std::vector< Feature::MinutiaPoint > BiometricEvaluation::Feature::Sort::sort (
            std::vector< Feature::MinutiaPoint > & minutia,
            const Kind & sortOrder)
```

Sort (p. 117) minutia.

Parameters

minutia	Minutia to be sorted.
sortOrder	Order in which to sort minutia.

Exceptions

Error::NotImplemented *(p. 636)*	sortOrder is not implemented.
Error::StrategyError *(p. 789)*	Center of mass is specified, but no minutia.

G.5.3.2 stableSort()

```
std::vector< Feature::MinutiaPoint > BiometricEvaluation::Feature::Sort::stableSort (
            std::vector< Feature::MinutiaPoint > & minutia,
            const Kind & sortOrder)
```

Sort (p. 117) minutia, maintaining existing order if elements are otherwise deemed equal.

Parameters

minutia	Minutia to be sorted.
sortOrder	Order in which to sort minutia.

Exceptions

Error::NotImplemented (p. 636)	sortOrder is not implemented.
Error::StrategyError (p. 789)	Center of mass is specified, but no minutia.

G.6 BiometricEvaluation::Finger Namespace Reference

Biometric information relating to finger images and derived information.

Classes

- class **AN2KMinutiaeDataRecord**

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

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

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

  A class to represent single finger view and derived information.
- class **ANSI2004View**

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

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

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

  A class to represent single finger view and derived information.

Typedefs

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

Enumerations

- enum class **PatternClassification** {
  PlainArch = 0 , **TentedArch** , **RadialLoop** , **UlnarLoop** ,
  PlainWhorl , **CentralPocketLoop** , **DoubleLoop** , **AccidentalWhorl** ,
  Whorl , **RightSlantLoop** , **LeftSlantLoop** , **Scar** ,
  Amputation , **Unknown** }
- enum class **Position** {
  Unknown = 0 , **RightThumb** = 1 , **RightIndex** = 2 , **RightMiddle** = 3 ,
  RightRing = 4 , **RightLittle** = 5 , **LeftThumb** = 6 , **LeftIndex** = 7 ,
  LeftMiddle = 8 , **LeftRing** = 9 , **LeftLittle** = 10 , **PlainRightThumb** = 11 ,
  PlainLeftThumb = 12 , **PlainRightFourFingers** = 13 , **PlainLeftFourFingers** = 14 , **LeftRight↩**
  Thumbs = 15 ,
  RightExtraDigit = 16 , **LeftExtraDigit** = 17 , **UnknownFrictionRidge** = 18 , **EJI** = 19 ,
  RightIndexMiddle = 40 , **RightMiddleRing** = 41 , **RightRingLittle** = 42 , **LeftIndexMiddle** = 43 ,
  LeftMiddleRing = 44 , **LeftRingLittle** = 45 , **RightIndexLeftIndex** = 46 , **RightIndexMiddleRing** =
  47 ,
  RightMiddleRingLittle = 48 , **LeftIndexMiddleRing** = 49 , **LeftMiddleRingLittle** = 50 , **Plain↩**
  RightFourTips = 51 ,
  PlainLeftFourTips = 52 , **PlainRightFiveTips** = 53 , **PlainLeftFiveTips** = 54 }

  Finger position codes.
- enum class **Impression** {
  PlainContact = 0 , **LiveScanPlain** = 0 , **RolledContact** = 1 , **LiveScanRolled** = 1 ,
  NonLiveScanPlain = 2 , **NonLiveScanRolled** = 3 , **LatentImage** = 4 , **LatentImpression** = 4 ,
  LatentTracing = 5 , **LatentPhoto** = 6 , **LatentLift** = 7 , **LiveScanSwipe** = 8 ,
  LiveScanVerticalSwipe = 8 , **LiveScanPalm** = 10 , **NonLiveScanPalm** = 11 , **LatentPalmImpression**
  = 12 ,
  LatentPalmTracing = 13 , **LatentPalmPhoto** = 14 , **LatentPalmLift** = 15 , **LiveScanOpticalContact↩**
  Plain = 20 ,
  LiveScanOpticalContactRolled = 21 , **LiveScanNonOpticalContactPlain** = 22 , **LiveScanNonOptical↩**
  ContactRolled = 23 , **ContactlessPlainStationarySubject** = 24 ,
  LiveScanOpticalContactlessPlain = 24 , **ContactlessRolledStationarySubject** = 25 , **LiveScan↩**
  OpticalContactlessRolled = 25 , **LiveScanNonOpticalContactlessPlain** = 26 ,
  LiveScanNonOpticalContactlessRolled = 27 , **Other** = 28 , **Unknown** = 29 , **ContactlessRolled↩**
  MovingSubject = 41 ,
  ContactlessPlainMovingSubject = 42 }
- enum class **FingerImageCode** {
  EJI = 0 , **RolledTip** , **FullFingerRolled** , **FullFingerPlainLeft** ,
  FullFingerPlainCenter , **FullFingerPlainRight** , **ProximalSegment** , **DistalSegment** ,
  MedialSegment , **NA** }
- enum class **CaptureTechnology** {
  Unknown = 0 , **Other** = 1 , **ScannedInkOnPaper** = 2 , **OpticalTIRBright** = 3 ,
  OpticalTIRDark = 4 , **OpticalDINative** = 5 , **OpticalDILowFrequenceyUnwrapped** = 6 , **Three↩**
  DimensionalHighFrequencyUnwrapped = 7 ,
  Capacitive = 9 , **CapacitiveRF** = 10 , **Electroluminescent** = 11 , **ReflectedUltrasonic** = 12 ,
  UltrasonicImpediography = 13 , **Thermal** = 14 , **DirectPressureSensitive** = 15 , **IndirectPressure** =
  16 ,

LiveTape = 17 , **LatentImpression** = 18 , **LatentPhoto** = 19 , **LatentMold** = 20 , **LatentTracing** = 21 , **LatentLift** = 22 }

Functions

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

  Output stream overload for FingerSegmentPosition.

G.6.1 Detailed Description

Biometric information relating to finger images and derived information.

The **Finger** (p. 122) package gathers all finger related matters, including classes to represent finger minutiae and helper functions for conversion between biometric representations. Contained within this namespace are classes to represent specific record formats, such as ANSI/NIST finger image records.

G.6.2 Enumeration Type Documentation

G.6.2.1 CaptureTechnology

`enum class` **`BiometricEvaluation::Finger::CaptureTechnology`** `[strong]`

Friction Ridge Capture Technology codes.

G.6.2.2 FingerImageCode

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

Joint and tip codes.

G.6.2.3 Impression

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

Finger (p. 122), palm, and latent impression types.

G.6.2.4 PatternClassification

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

Pattern classification codes.

G.6.2.5 Position

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

Finger (p. 122) position codes.

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

G.7 BiometricEvaluation::Framework Namespace Reference

Information about the framework.

Classes

- class **API**

  *A convenient way to execute biometric technology evaluation **API** (p. 286) methods safely.*
- class **Status**

Enumerations

- enum class **APICurrentState** {
  NeverCalled , **WatchdogExpired** , **SignalCaught** , **ExceptionCaught** ,
  Running , **Completed** }

Functions

- unsigned int **getMajorVersion** ()

  Framework *(p. 124) major version.*
- unsigned int **getMinorVersion** ()

  Framework *(p. 124) minor version.*
- std::string **getCompiler** ()

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

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

  Time *(p. 185) when this framework was compiled.*
- std::string **getCompilerVersion** ()

  Version string of compiler used to compile this framework.
- std::string **to_string** (const **Status** &status)

  Obtain a textual representation of a ***Status*** *(p. 786).*
- std::ostream & **operator<<** (std::ostream &s, const **Status** &status)

  Output stream operator overload.

G.7.1 Detailed Description

Information about the framework.

G.7.2 Enumeration Type Documentation

G.7.2.1 APICurrentState

`enum class` **`BiometricEvaluation::Framework::APICurrentState`** `[strong]`

Reasons operations could not complete.

Enumerator

NeverCalled	Operation was never executed.
WatchdogExpired	Watchdog timer expired.

Enumerator

SignalCaught	Signal handler was invoked.
ExceptionCaught	An exception was caught.
Running	Operation is running.
Completed	Operation has returned.

G.7.3 Function Documentation

G.7.3.1 getCompileDate()

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

Date when this framework was compiled.

Returns

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

G.7.3.2 getCompiler()

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

Compiler used to compile this framework.

Returns

The name of the compiler used to compile this framework.

G.7.3.3 getCompilerVersion()

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

Version string of compiler used to compile this framework.

Returns

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

G.7.3.4 getCompileTime()

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

Time (p. 185) when this framework was compiled.

Returns

Time (p. 185) when this framework was compiled, in the form "HH:MM:SS"

G.7.3.5 getMajorVersion()

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

Framework (p. 124) major version.

Returns

The major version number of the BiometricFramework

G.7.3.6 getMinorVersion()

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

Framework (p. 124) minor version.

Returns

The minor version of the **BiometricEvaluation** (p. 111) framework.

G.7.3.7 operator<<()

```
std::ostream & BiometricEvaluation::Framework::operator<< (
            std::ostream & s,
            const  Status & status)
```

Output stream operator overload.

Parameters

s	Output stream.
status	**Status** (p. 786) object to output.

Returns

s appended with string representation of status.

G.7.3.8 to_string()

```
std::string BiometricEvaluation::Framework::to_string (
            const  Status & status)
```

Obtain a textual representation of a **Status** (p. 786).

Parameters

status	**Status** (p. 786) object to convert.

Returns

Textual representation of status.

G.8 BiometricEvaluation::Image Namespace Reference

Basic information relating to images.

Classes

- class **BMP**

  A BMP-encoded image.
- struct **Coordinate**

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

  Represent attributes common to all images.
- class **JPEG**

  A JPEG-encoded image.
- class **JPEG2000**

  A JPEG-2000-encoded image.
- class **JPEGL**

  A Lossless JPEG-encoded image.
- class **NetPBM**

  A NetPBM-encoded image.
- class **PNG**

  A PNG-encoded image.
- class **Raw**

  An image with no encoding or compression.
- struct **Resolution**

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

  *A structure to represent a region of interest (**ROI** (p. 746)), which is a bounding box and a set of coordinates.*
- struct **Size**

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

  A WSQ-encoded image.

Typedefs

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

Enumerations

- enum class **CompressionAlgorithm** {
  None = 0 , **Facsimile** = 1 , **WSQ20** = 2 , **JPEGB** = 3 ,
  JPEGL = 4 , **JP2** = 5 , **JP2L** = 6 , **PNG** = 7 ,
  NetPBM = 8 , **BMP** = 9 , **TIFF** = 10 }
- enum class **PixelFormat** { **MonoWhite** = 0 , **MonoBlack** = 1 , **Gray8** = 2 , **RGB24** = 3 }

Functions

- std::string **to_string** (const **Coordinate** &c)

  *Convert **Coordinate** (p. 377) to std::string.*
- std::ostream & **operator**<< (std::ostream &, const **Coordinate** &)
- bool **operator==** (const **Coordinate** &lhs, const **Coordinate** &rhs)
- bool **operator!=** (const **Coordinate** &lhs, const **Coordinate** &rhs)
- std::string **to_string** (const CoordinateSet &coordinates)

  Convert CoordinateSet to std::string.
- std::ostream & **operator**<< (std::ostream &stream, const CoordinateSet &coordinates)

  Output stream overload for CoordinateSet.
- std::string **to_string** (const **Size** &s)

  *Convert **Size** (p. 763) to std::string.*
- std::ostream & **operator**<< (std::ostream &, const **Size** &)
- bool **operator==** (const **Size** &lhs, const **Size** &rhs)
- bool **operator!=** (const **Size** &lhs, const **Size** &rhs)
- std::string **to_string** (const **Resolution** &r)

  *Convert **Resolution** (p. 737) to std::string.*
- std::ostream & **operator**<< (std::ostream &, const **Resolution** &)
- bool **operator==** (const **Resolution** &lhs, const **Resolution** &rhs)
- bool **operator!=** (const **Resolution** &lhs, const **Resolution** &rhs)
- float **distance** (const **Coordinate** &p1, const **Coordinate** &p2)

  Calculate the distance between two points.
- **BiometricEvaluation::Memory::uint8Array removeComponents** (const **BiometricEvaluation::↩ Memory::uint8Array** &rawData, const uint8_t bitDepth, const std::vector< bool > &components)

  Remove components from a decompressed image's raw byte representation.
- std::string **to_string** (const **ROI** &r)

  *Convert **ROI** (p. 746) to std::string.*
- std::ostream & **operator**<< (std::ostream &, const **ROI** &)
- bool **operator==** (const **ROI** &lhs, const **ROI** &rhs)
- bool **operator!=** (const **ROI** &lhs, const **ROI** &rhs)

Variables

- const double **CentimetersPerInch** = 2.54
- const double **MillimetersPerInch** = **CentimetersPerInch** ∗ 10

G.8.1 Detailed Description

Basic information relating to images.

Classes and methods for manipulating images.

The **Image** (p. 477) package gathers all image related matters, including classes to represent an image, coordinates, and functions for conversion between biometric representations.

G.8.2 Enumeration Type Documentation

G.8.2.1 CompressionAlgorithm

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

Image (p. 477) compression algorithms.

G.8.2.2 PixelFormat

`enum class` **`BiometricEvaluation::Image::PixelFormat`** `[strong]`

Image (p. 477) pixel formats.

Enumerator

MonoWhite	1 bit/pixel, 0 is white, 1 = black
MonoBlack	1 bit/pixel, 0 is black, 1 = white
Gray8	8-bit gray
RGB24	8-bit red/8-bit blue/8-bit green

G.8.3 Function Documentation

G.8.3.1 distance()

```
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.

G.8.3.2 operator<<()

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

Output stream overload for CoordinateSet.

Parameters

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

Returns

stream with a coordinates textual representation appended.

G.8.3.3 removeComponents()

```
BiometricEvaluation::Memory::uint8Array BiometricEvaluation::Image::removeComponents (
            const  BiometricEvaluation::Memory::uint8Array & rawData,
            const uint8_t bitDepth,
            const std::vector< bool > & components)
```

Remove components from a decompressed image's raw byte representation.

Parameters

in	*rawData*	**Raw** (p. 688) byte representation of an image.
in	*bitDepth*	The number of bits that represents a single component in raw↩ Data (only 8 and 16 are supported).

Parameters

in	*components*	A bitset representing the components of the image, where true values represent components to be removed. For example, in a 4-component image where fourth component should be removed, this parameter would be {false, false, false, true}.

Returns

Copy of `rawData` with true `components` removed.

Exceptions

BiometricEvaluation::Error::ParameterError (p. 655)	Invalid `bitDepth` parameter
BiometricEvaluation::Error::StrategyError (p. 789)	`rawData` does not appear to be sized large enough for the `bitsPerC`

G.8.3.4 to_string() [1/5]

```
std::string BiometricEvaluation::Image::to_string (
            const  Coordinate & c)
```

Convert **Coordinate** (p. 377) to std::string.

Parameters

c	**Coordinate** (p. 377) to convert to std::string.

Returns

std::string representation of c.

G.8.3.5 to_string() [2/5]

```
std::string BiometricEvaluation::Image::to_string (
            const CoordinateSet & coordinates)
```

Convert CoordinateSet to std::string.

Parameters

coordinates	CoordinateSet to convert to std::string.

Returns

std::string representation of coordinates.

G.8.3.6 to_string() [3/5]

```
std::string BiometricEvaluation::Image::to_string (
            const  Resolution & r)
```

Convert **Resolution** (p. 737) to std::string.

Parameters

r	**Resolution** (p. 737) to convert to std::string.

Returns

std::string representation of r.

G.8.3.7 to_string() [4/5]

```
std::string BiometricEvaluation::Image::to_string (
            const  ROI & r)
```

Convert **ROI** (p. 746) to std::string.

Parameters

r	**ROI** (p. 746) to convert to std::string.

Returns

std::string representation of r.

G.8.3.8 to_string() [5/5]

```
std::string BiometricEvaluation::Image::to_string (
            const  Size & s)
```

Convert **Size** (p. 763) to std::string.

Parameters

s	**Size** (p. 763) to convert to std::string.

Returns

std::string representation of s.

G.8.4 Variable Documentation

G.8.4.1 CentimetersPerInch

```
const double BiometricEvaluation::Image::CentimetersPerInch = 2.54
```

Number of centimeters in one inch

G.8.4.2 MillimetersPerInch

```
const double BiometricEvaluation::Image::MillimetersPerInch = CentimetersPerInch * 10
```

Number of millimeters in one inch

G.9 BiometricEvaluation::IO Namespace Reference

Input/Output functionality.

Namespaces

- namespace **Utility**

Classes

- class **ArchiveRecordStore**

  *This class implements the **IO::RecordStore** (p. 700) interface by storing data items in single file, with an associated manifest file.*
- class **AutoLogger**

  *The **AutoLogger** (p. 327) class provides an interface for writing to a log file within a background thread. The content for log entries is retrieved via a call back to the owning object.*
- class **CompressedRecordStore**

  *Sibling-implemented **IO::RecordStore** (p. 700) with Compression.*
- class **Compressor**

  Common interface for classes providing compressing and decompressing functionality.
- class **DBRecordStore**

  *A class that implements **IO::RecordStore** (p. 700) using a Berkeley DB database as the underlying record storage system.*
- class **FileLogCabinet**

  A class to represent a collection of log sheets.
- class **FileLogsheet**

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

  A class to represent the record store data storage mechanism implemented as files for each record.
- class **GZip**

  *An **IO::Compressor** (p. 360) for gzip compression from zlib.*
- class **ListRecordStore**

  ***IO::RecordStore** (p. 700) that reads a list of keys from a text file, and retrieves the data from another **IO::RecordStore** (p. 700).*
- class **Logsheet**

  A class to represent a logging mechanism.
- class **PersistentRecordStoreUnion**

*An implementation of **IO::RecordStoreUnion** (p. 729) that persists across instantiations.*

- class **Properties**

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

  *An **IO::Properties** (p. 674) object persisted in a file on disk.*
- class **RecordStore**

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

  Generic ForwardIterator for all RecordStores.
- class **RecordStoreUnion**

  A collection of N related read-only RecordStores, operated on simultaneously.
- class **SQLiteRecordStore**

  *An **IO::RecordStore** (p. 700) implementation using a SQLite database as the underlying record storage system.*
- class **SysLogsheet**

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

Enumerations

- enum class **Mode** { **ReadWrite** = 0 , **ReadOnly** = 1 }

G.9.1 Detailed Description

Input/Output functionality.

The **IO** (p. 136) package contains classes and functions used to abstract input and output operations and provide for robust error handling on behalf of the application.

G.9.2 Enumeration Type Documentation

G.9.2.1 Mode

`enum class` **`BiometricEvaluation::IO::Mode`** `[strong]`

Accessibility of object.

Enumerator

ReadWrite	Constant indicating the state of an object that manages some underlying file is accessible for reading and writing.
ReadOnly	Constant indicating the state of an object that manages some underlying file is accessible for reading only.

G.10 BiometricEvaluation::IO::Utility Namespace Reference

Functions

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

Remove a directory using directory name and parent pathname.

- void **removeDirectory** (const std::string &pathname)

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

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

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

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

  Read the contents of a file into an 8-bit AutoArray.
- void **writeFile** (const uint8_t *data, const size_t size, const std::string &path, std::ios_base::openmode mode=std::ios_base::binary)

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

  Write the contents of an 8-bit AutoArray to a file.
- void **readPipe** (void *data, size_t size, int pipeFD)

  Read from an open pipe into a buffer.
- void **readPipe** (**Memory::uint8Array** &data, int pipeFD)

  Read from an open pipe into an 8-bit AutoArray.
- void **writePipe** (const void *data, size_t size, int pipeFD)

  Write the contents of a buffer to a pipe.
- void **writePipe** (const **Memory::uint8Array** &data, int pipeFD)

  Write the contents of an 8-bit AutoArray to a pipe.
- bool **isReadable** (const std::string &pathname)

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

  Determine if the real user has write access permissions to this file.
- std::string **createTemporaryFile** (const std::string &prefix="", const std::string &parentDir="/tmp")

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

  Create a temporary file.
- uint64_t **countLines** (const std::string &path)

  Count the number of newline characters in a text file.
- uint64_t **countLines** (const **Memory::uint8Array** &textBuffer)

  Count the number of newline characters in a buffer of a text file.

G.10.1 Detailed Description

A class containing utility functions used for **IO** (p. 136) operations. These functions are class methods.

G.10.2 Function Documentation

G.10.2.1 copyDirectoryContents()

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

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

Parameters

in	*sourcepath*	The name of the directory whose contents are to be moved.
in	*targetpath*	The name of the directory where the contents of the sourcepath are to be moved.

Parameters

in	*removesource*	Flag indicating whether to remove the source directory after the copy is complete.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The source named directory does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system, or the directoy name or prefi

G.10.2.2 countLines() [1/2]

```
uint64_t BiometricEvaluation::IO::Utility::countLines (
            const  Memory::uint8Array & textBuffer)
```

Count the number of newline characters in a buffer of a text file.

Parameters

path	Buffer of text file that has been read in.

Returns

Number of newline characters in buffer.

G.10.2.3 countLines() [2/2]

```
uint64_t BiometricEvaluation::IO::Utility::countLines (
            const std::string & path)
```

Count the number of newline characters in a text file.

Parameters

path	Path to text file.

Returns

Number of newline characters in file at path.

Exceptions

Error::FileError (p. 420)	Could not open path.

G.10.2.4 createTemporaryFile() [1/2]

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

Create a temporary file.

Parameters

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

Exceptions

Error::FileError (p. 420)	Could not create or close temporary file.
Error::MemoryError (p. 604)	**Error** (p. 112) 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.

The default argument for `parentDir` is "/tmp". On Windows, this will be replaced with the appropriate Windows temporary directory, but Windows does not guarantee write access.

G.10.2.5 createTemporaryFile() [2/2]

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

Create a temporary file.
Exclusivity to the file stream is guaranteed.

Parameters

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

Exceptions

Error::FileError (p. 420)	Could not create or close temporary file.
Error::MemoryError (p. 604)	**Error** (p. 112) allocating memory for file name.

Returns

Open file stream to path.

Note

Caller must fclose(3) the returned stream.

The default argument for `parentDir` is "/tmp". On Windows, this will be replaced with the appropriate Windows temporary directory, but Windows does not guarantee write access.

G.10.2.6 fileExists()

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

Indicate whether a file exists.

Parameters

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

Returns

true if the file exists, false otherwise.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system, or pathname is malformed.

G.10.2.7 getFileSize()

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

Get the size of a file.

Parameters

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

Returns

The file size.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named directory does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system, or pathname is malformed.

G.10.2.8 isReadable()

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

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

Parameters

in	*pathname*	Path to the file to check.

Returns

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

Warning

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

See also

BiometricEvaluation::IO::Utility::fileExists() (p. 145)

G.10.2.9 isWritable()

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

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

Parameters

in	*pathname*	Path to the file to check.

Returns

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

Warning

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

See also

BiometricEvaluation::IO::Utility::fileExists() (p. 145)

G.10.2.10 makePath()

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

Create an entire directory tree.

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

Parameters

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

Returns

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

G.10.2.11 readFile()

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

Read the contents of a file into an 8-bit AutoArray.

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 an AutoArray.

Exceptions

Error::ObjectDoesNotExist (p. 637)	path does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

G.10.2.12 readPipe() [1/2]

```
void BiometricEvaluation::IO::Utility::readPipe (
             Memory::uint8Array & data,
            int pipeFD)
```

Read from an open pipe into an 8-bit AutoArray.

Wraps the read(2) system call by reading the requested amount of data from a pipe file descriptor, ∗ handling all errors and signals.

Parameters

data	Data array to read into.

Parameters

pipeFD	The file descriptor of the pipe.

Exceptions

ObjectDoesNotExist	The reading end of the pipe has been closed.
FileError	The data could not be written in the entirety; **Error::errorStr()** (p. 113) may contain more information.

G.10.2.13 readPipe() [2/2]

```
void BiometricEvaluation::IO::Utility::readPipe (
            void * data,
            size_t size,
            int pipeFD)
```

Read from an open pipe into a buffer.

Wraps the read(2) system call by reading the requested amount of data from a pipe file descriptor, handling all errors and signals.

Parameters

data	Data buffer to store the data being read.
size	Size of data to read.
pipeFD	The file descriptor of the pipe.

Exceptions

ObjectDoesNotExist	The writing end of the pipe has been closed.
FileError	The data could not be written in the entirety; **Error::errorStr()** (p. 113) may contain more information.

G.10.2.14 removeDirectory() [1/2]

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

Remove a directory using directory name and parent pathname.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named directory does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system, or the directoy name or prefi

G.10.2.15 removeDirectory() [2/2]

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

Remove a directory using a complete pathname.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named directory does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system, or the path name is malforme

G.10.2.16 setAsideName()

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

Set aside a file or directory name.

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

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named object does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system, the name or prefix is malform

G.10.2.17 sumDirectoryUsage()

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

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

Parameters

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

Returns

The sum of file and directory sizes.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named directory does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system, or pathname is malformed.

G.10.2.18 writeFile() [1/2]

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

Write the contents of an 8-bit AutoArray to a file.

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

Parameters

data	Data array to write.
path	Path to file to create with contents of data.

Parameters

mode	Bitwise OR'd arguments to send to the file stream constructor.

Exceptions

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

G.10.2.19 writeFile() [2/2]

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

Write the contents of a buffer to a file.

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

Parameters

data	Data buffer to write.
size	Size of data.
path	Path to file to create with contents of data.

Parameters

mode	Bitwise OR'd arguments to send to the file stream constructor.

Exceptions

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

G.10.2.20 writePipe() [1/2]

```
void BiometricEvaluation::IO::Utility::writePipe (
            const  Memory::uint8Array & data,
            int pipeFD)
```

Write the contents of an 8-bit AutoArray to a pipe.

Wraps the write(2) system call by writing all data to a pipe file descriptor, handling all errors and signals.

Parameters

data	Data array to write.
pipeFD	The file descriptor of the pipe.

Exceptions

ObjectDoesNotExist	The reading end of the pipe has been closed.
FileError	The data could not be written in the entirety; **Error::errorStr()** (p. 113) may contain more information.

G.10.2.21 writePipe() [2/2]

```
void BiometricEvaluation::IO::Utility::writePipe (
            const void * data,
            size_t size,
            int pipeFD)
```

Write the contents of a buffer to a pipe.

Wraps the write(2) system call by writing all data to a pipe file descriptor, handling all errors and signals.

Parameters

data	Data buffer to write.
size	Size of data.
pipeFD	The file descriptor of the pipe.

Exceptions

ObjectDoesNotExist	The reading end of the pipe has been closed.
FileError	The data could not be written in the entirety; **Error::errorStr()** (p. 113) may contain more information.

G.11 BiometricEvaluation::Iris Namespace Reference

Biometric information relating to iris images and derived information.

Classes

- class **INCITSView**

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

  A class to represent single iris view and derived information.

Enumerations

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

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

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

Iris image type classification codes.

- enum class **Orientation** { **Undefined** = 0 , **Base** = 1 , **Flipped** = 2 }

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

  Iris image compression type.
- enum class **CameraRange** { **Unassigned** = 0 , **Failed** = 1 , **Overflow** = 2 }

  Range from camera lens center to subject iris.

G.11.1 Detailed Description

Biometric information relating to iris images and derived information.

The **Iris** (p. 155) package gathers all iris related matters, including classes to represent iris information and helper functions for conversion between biometric representations. Contained within this namespace are classes to represent specific record formats, such as ISO 19794-6.

G.12 BiometricEvaluation::Memory Namespace Reference

Support for memory-related operations.

Namespaces

- namespace **AutoArrayUtility**

Classes

- class **AutoArray**

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

  *RandomAccessIterator for any **AutoArray** (p. 306).*
- class **AutoBuffer**
- class **IndexedBuffer**

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

  Define a type that is visible when T is not an array.
- struct **unique_if**< **T[ ]**>

  Define a type that is visible when T is an unknown-bound array.
- struct **unique_if**< **T[S]**>

  Define a type that is visible when T is an known-bound array.

Typedefs

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

Functions

- template<typename T , typename... Ts>
  unique_if< T >::unique_single **make_unique** (Ts &&... params)

  Framework *(p. 124) version of std::make_unique for non-array types.*
- template<class T >
  unique_if< T >::unique_array_unknown_bound **make_unique** (size_t size)

  Framework *(p. 124) version of std::make_unique for unknown-bound arrays.*
- template<class T , class... Ts>
  unique_if< T >::unique_array_known_bound **make_unique** (Ts &&...)=delete

  Framework *(p. 124) version of std::make_unique for known-bound arrays.*
- bool **isLittleEndian** ()

  Determine endianess of current platform.
- template<typename T >
  bool **operator==** (const **AutoArray**< T > &lhs, const **AutoArray**< T > &rhs)
- template<typename T >
  bool **operator!=** (const **AutoArray**< T > &lhs, const **AutoArray**< T > &rhs)
- template<typename T >
  bool **operator**< (const **AutoArray**< T > &lhs, const **AutoArray**< T > &rhs)
- template<typename T >
  bool **operator**<= (const **AutoArray**< T > &lhs, const **AutoArray**< T > &rhs)
- template<typename T >
  bool **operator**> (const **AutoArray**< T > &lhs, const **AutoArray**< T > &rhs)
- template<typename T >
  bool **operator**>= (const **AutoArray**< T > &lhs, const **AutoArray**< T > &rhs)

G.12.1 Detailed Description

Support for memory-related operations.

The **Memory** (p. 156) package contains templates and classes that are used to manage memory, auto-sizing arrays, for example.

G.12.2 Function Documentation

G.12.2.1 isLittleEndian()

```
bool BiometricEvaluation::Memory::isLittleEndian ()  [inline]
```

Determine endianess of current platform.

Returns

true if current platform is little endian. false otherwise.

G.12.2.2 make_unique() [1/3]

```
template<class T >
unique_if< T >::unique_array_unknown_bound BiometricEvaluation::Memory::make_unique (
            size_t size)
```

Framework (p. 124) version of std::make_unique for unknown-bound arrays.

Note

Coming in C++14. This implementation is taken from the LLVM implementation.

This function shall not participate in overload resolution unless T is an array of unknown bound.

G.12.2.3 make_unique() [2/3]

```
template<typename T , typename...  Ts>
unique_if< T >::unique_single BiometricEvaluation::Memory::make_unique (
            Ts &&...  params)
```

Framework (p. 124) version of std::make_unique for non-array types.

Note

Coming in C++14. This implementation is taken from "Effective Modern C++" by Scott Meyers, modified to participate in the overload resolution only when T is not an array.

This function shall not participate in overload resolution unless T is not an array.

G.12.2.4 make_unique() [3/3]

```
template<class T , class...  Ts>
unique_if< T >::unique_array_known_bound BiometricEvaluation::Memory::make_unique (
            Ts && ...)  [delete]
```

Framework (p. 124) version of std::make_unique for known-bound arrays.

Note

Coming in C++14. This implementation is taken from the LLVM implementation.

This function shall not participate in overload resolution unless T is an array of known bound.

G.12.2.5 operator"!=()

```
template<typename T >
bool BiometricEvaluation::Memory::operator!= (
            const  AutoArray< T > & lhs,
            const  AutoArray< T > & rhs)
```

Returns

Whether size or any accessible entries differ.

G.12.2.6 operator<()

```
template<typename T >
bool BiometricEvaluation::Memory::operator< (
            const  AutoArray< T > & lhs,
            const  AutoArray< T > & rhs)
```

Returns

Lexicographical comparison of accessible entries.

G.12.2.7 operator<=()

```
template<typename T >
bool BiometricEvaluation::Memory::operator<= (
            const  AutoArray< T > & lhs,
            const  AutoArray< T > & rhs)
```

Returns

Lexicographical comparison of accessible entries.

G.12.2.8 operator==()

```
template<typename T >
bool BiometricEvaluation::Memory::operator== (
            const  AutoArray< T > & lhs,
            const  AutoArray< T > & rhs)
```

Returns

Equivalence of all accessible entries and size.

G.12.2.9 operator>()

```
template<typename T >
bool BiometricEvaluation::Memory::operator> (
            const  AutoArray< T > & lhs,
            const  AutoArray< T > & rhs)
```

Returns

Lexicographical comparison of accessible entries.

G.12.2.10 operator>=()

```
template<typename T >
bool BiometricEvaluation::Memory::operator>= (
            const  AutoArray< T > & lhs,
            const  AutoArray< T > & rhs)
```

Returns

Lexicographical comparison of accessible entries.

G.13 BiometricEvaluation::Memory::AutoArrayUtility Namespace Reference

Functions

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

  *Cast an **AutoArray** (p. 306) of uint8_t or char to a char*.*

- template<typename T , typename = typename std::enable_if<std::is_same<T, uint8_t>::value || std::is_same<T, char>::value>::type>
  std::string **getString** (const **AutoArray**< T > &aa, typename **AutoArray**< T >::size_type count)
  *Convert a uint8_t or char **AutoArray** (p. 306) to a string.*
- template<typename T , typename = typename std::enable_if<std::is_same<T, uint8_t>::value || std::is_same<T, char>::value>::type>
  void **setString** (**AutoArray**< T > &aa, const std::string &str)
  Copy a string into an AutoAray of uint8_t or char.
- template<typename T , typename = typename std::enable_if<std::is_same<T, uint8_t>::value || std::is_same<T, char>::value>::type>
  void **setString** (**AutoArray**< T > &aa, const char *str,...)
  Copy a string into an AutoAray of uint8_t or char.

G.13.1 Detailed Description

Convenience functions for AutoArrays.

G.13.2 Function Documentation

G.13.2.1 cstr()

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

Cast an **AutoArray** (p. 306) of uint8_t or char to a char*.

Parameters

rahc	**AutoArray** (p. 306) to cast.

Returns

rahc casted as a char*.

G.13.2.2 getString()

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

Convert a uint8_t or char **AutoArray** (p. 306) to a string.

Parameters

aa	**AutoArray** (p. 306) to stringify.
count	Last byte of aa to include in the returned string.

Returns

String representation of aa.

Exceptions

Error::MemoryError *(p. 604)*	count > aa.size()

G.13.2.3 setString() [1/2]

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

Copy a string into an AutoAray of uint8_t or char.

Parameters

aa	**AutoArray** (p. 306) whose contents will be replaced with str.

Parameters

str	printf-style format string.
...	Variable list of arguments for printf formatting.

G.13.2.4 setString() [2/2]

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

Copy a string into an AutoAray of uint8_t or char.

Parameters

aa	**AutoArray** (p. 306) whose contents will be replaced with str.
str	String to assign to **AutoArray** (p. 306).

G.14 BiometricEvaluation::MPI Namespace Reference

Common declarations and functions for the MPI-based functionality.

Classes

- class **CSVDistributor**

  An implementation of the MPI::Distrbutor abstraction that distribute lines of a text file via work packages.
- class **CSVProcessor**

  An implementation of a work package processor that will extract lines (and optionally tokenize) a line from a CSV text file.
- class **CSVResources**
- class **Distributor**

  *A class to represent an **MPI** (p. 162) task that distributes work to other tasks.*
- class **Exception**
- class **Receiver**

  *A class to represent an **MPI** (p. 162) task that receives WorkPackages containers from the **Distributor** (p. 405).*
- class **RecordProcessor**

  *An implementation of a work package processor that will extract record store keys, and optionally, values, from a **WorkPackage** (p. 841).*
- class **RecordStoreDistributor**

  An implementation of the MPI::Distrbutor abstraction that uses a record store for input to create the work packages.
- class **RecordStoreResources**

  *A class to represent a set of resources needed by an **MPI** (p. 162) program using a RecordStore for input.*
- class **Resources**
- class **Runtime**

  ***Runtime** (p. 748) support for the startup/shutdown of **MPI** (p. 162) jobs.*
- class **TerminateJob**

  *An exception that when thrown from a Task should result in the entire job (all tasks) being shut down by the **Distributor** (p. 405).*
- class **WorkPackage**

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

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

Typedefs

- using **taskcmd_t** = std::underlying_type< **TaskCommand**>::type
- using **taskstat_t** = std::underlying_type< **TaskStatus**>::type
- using **msgtag_t** = std::underlying_type< **MessageTag**>::type

Enumerations

- enum class **TaskCommand** : int32_t {
  Continue = 0 , **Ignore** = 1 , **Exit** = 2 , **QuickExit** = 3 ,
  TermExit = 4 }
- enum class **TaskStatus** : int32_t { **OK** = 0 , **Failed** = 1 , **Exit** = 2 , **RequestJobTermination** = 3 }
- enum class **MessageTag** : int32_t { **Control** = 0 , **Data** = 1 , **OOB** = 2 }

Functions

- std::string **generateUniqueID** ()

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

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

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

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

  *Open a Logsheet object for a component of the **MPI** (p. 162) framework.*

Variables

- bool **Exit**
- bool **QuickExit**
- bool **TermExit**
- bool **checkpointEnable**
- bool **doCheckpointRestore**

G.14.1 Detailed Description

Common declarations and functions for the MPI-based functionality.

G.14.2 Typedef Documentation

G.14.2.1 msgtag_t

`using` **`BiometricEvaluation::MPI::msgtag_t`** `= std::underlying_type<` **`MessageTag`**`>::type`

Storage type for MessageTag.

G.14.2.2 taskcmd_t

`using` **`BiometricEvaluation::MPI::taskcmd_t`** `= std::underlying_type<` **`TaskCommand`**`>::type`

Storage type for TaskCommand.

G.14.2.3 taskstat_t

`using` **`BiometricEvaluation::MPI::taskstat_t`** `= std::underlying_type<` **`TaskStatus`**`>::type`

Storage type for TaskStatus.

G.14.3 Enumeration Type Documentation

G.14.3.1 MessageTag

`enum class` **`BiometricEvaluation::MPI::MessageTag`** `: int32_t [strong]`

The types of messages sent between **MPI** (p. 162) task processes.

Enumerator

Control	A control message (start, exit, etc.)
Data	A data message.
OOB	An out-of-band message, used when the normal control/-data messaging cannot be used.

G.14.3.2 TaskCommand

`enum class` **`BiometricEvaluation::MPI::TaskCommand`** `: int32_t [strong]`

The command given to an **MPI** (p. 162) task.

Enumerator

Continue	Normal operation.
Ignore	Ignore the message.
Exit	Transition to the normal shutdown state.

Enumerator

QuickExit	Transition to the quick shut-down state.
TermExit	Transition to the im-med-itate shut-down state.

G.14.3.3 TaskStatus

`enum class` **`BiometricEvaluation::MPI::TaskStatus`** `: int32_t [strong]`

The status of an **MPI** (p. 162) distributor or receiver task.

Enumerator

OK	Normal opera-tion.
Failed	Failed to com-plete an opera-tion.
Exit	Transitioned to the shut-down state.
RequestJobTermination	Requesting that **Dis-trib-utor** (p. 405) stops the job.

G.14.4 Function Documentation

G.14.4.1 generateUniqueID()

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

Obtain a unique ID for the current process.

The ID is a string that is based on the host name, **MPI** (p. 162) rank, and process ID, formatted in a manner that can be used to uniquely name files.

Returns

The unique ID for the process.

G.14.4.2 logEntry()

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

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

Log messages may be streamed into the Logsheet and written as debug messages to aid tracing. In order to prevent log errors interfering with the **MPI** (p. 162) job, errors are managed, and therefore, log messages may stop if the Logsheet has failed.

Parameters

in	*logsheet*	The open Logsheet to write into.

G.14.4.3 logMessage()

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

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

In order to prevent log errors interfering with the **MPI** (p. 162) job, errors are managed, and therefore, log messages may stop if the Logsheet has failed.

Parameters

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

G.14.4.4 openLogsheet()

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

Open a Logsheet object for a component of the **MPI** (p. 162) framework.

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

Parameters

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

Returns

Shared pointer to the Logsheet object.

Exceptions

Error::ParameterError *(p. 655)*	Invalid URL.
Error::Exception *(p. 412)*	Failed to create the Logsheet object. The exception string will contain more information.

G.14.4.5 printStatus()

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

Print a status message to stdout.

Parameters

in	*message*	The messasge to be printed.

G.15 BiometricEvaluation::Palm Namespace Reference

Biometric information relating to palm images and derived information.

Classes

- class **AN2KView**

  *A class to represent a single **Palm** (p. 169) view and derived information.*

Enumerations

- enum class **Position** {
  Unknown = 20 , **RightFull** = 21 , **RightWriters** = 22 , **LeftFull** = 23 ,
  LeftWriters = 24 , **RightLower** = 25 , **RightUpper** = 26 , **LeftLower** = 27 ,
  LeftUpper = 28 , **RightOther** = 29 , **LeftOther** = 30 , **RightInterdigital** = 31 ,
  RightThenar = 32 , **RightHypothenar** = 33 , **LeftInterdigital** = 34 , **LeftThenar** = 35 ,
  LeftHypothenar = 36 , **RightGrasp** = 37 , **LeftGrasp** = 38 , **RightCarpelDelta** = 81 ,
  LeftCarpelDelta = 82 , **RightFullWithWriters** = 83 , **LeftFullWithWriters** = 84 , **RightWrist**↩
  Bracelet = 85 ,
  LeftWristBracelet = 86 }

  Palm position codes.

G.15.1 Detailed Description

Biometric information relating to palm images and derived information.
The **Palm** (p. 169) package gathers all palm related matters,

G.15.2 Enumeration Type Documentation

G.15.2.1 Position

`enum class` **`BiometricEvaluation::Palm::Position`** `[strong]`

Palm (p. 169) position codes.
These codes match those in ANSI/NIST. Other data formats may have to map codes into this set.

G.16 BiometricEvaluation::Plantar Namespace Reference

Biometric information relating to plantar images and derived information.

Enumerations

- enum class **Position** {
  UnknownSole = 60 , **RightSole** = 61 , **LeftSole** = 62 , **UnknownToe** = 63 ,
  RightBigToe = 64 }

  Plantar position codes.

G.16.1 Detailed Description

Biometric information relating to plantar images and derived information.

G.16.2 Enumeration Type Documentation

G.16.2.1 Position

`enum class` **BiometricEvaluation::Plantar::Position** `[strong]`

Plantar (p. 169) position codes.

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

G.17 BiometricEvaluation::Process Namespace Reference

Process (p. 170) information and controls.

Classes

- class **CommandCenter**
- class **CommandParser**
- class **ForkManager**

  Manager (p. *596*) *implementation that starts Workers by calling fork(2).*
- class **ForkWorkerController**

  Wrapper of a ***Worker*** (p. *828*) *returned from a* ***Process::ForkManager*** (p. *447*).
- class **Manager**

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

  Receives message from a client, forwarding to the central ***MessageCenter*** (p. *607*).
- class **POSIXThreadManager**

  Manager (p. *596*) *implementation that starts Workers in POSIX threads.*
- class **POSIXThreadWorkerController**

  Decorated ***Worker*** (p. *828*) *returned from a* ***Process::POSIXThreadManager*** (p. *666*).
- class **Semaphore**

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

  The ***Statistics*** (p. *780*) *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*** (p. *828*) *returned from a* ***Process::Manager*** (p. *596*).

Typedefs

- using **ParameterList**

G.17.1 Detailed Description

Process (p. 170) information and controls.

The **Process** (p. 170) package gathers all process related matters, including a class to obtain resource usage statistics.

G.17.2 Typedef Documentation

G.17.2.1 ParameterList

`using` **`BiometricEvaluation::Process::ParameterList`**

Initial value:

```
std::map<std::string, std::shared_ptr<void>>
```

Convenience alias for parameter lists to child routines

G.18 BiometricEvaluation::System Namespace Reference

Operating system, hardware, etc.

Classes

- class **MemoryLogger**

Functions

- uint32_t **getCPUCount** ()

  Obtain the number of central processing units that are online. Typically, this is the total logical CPU count for the system, often called hyperthreads.
- uint32_t **getCPUCoreCount** ()

  Obtain the number of CPU cores that are online.
- uint32_t **getCPUSocketCount** ()

  Obtain the number of CPU sockets that are online.
- uint64_t **getRealMemorySize** ()

  Obtain the amount of real memory in the system.
- std::map< std::string, uint64_t > **getMemInfo** ()
- double **getLoadAverage** ()

  Obtain the system load average for the last minute.

G.18.1 Detailed Description

Operating system, hardware, etc.

The **System** (p. 171) package gathers all system related matters, such as the operating system name, number of CPUs, etc.

G.18.2 Function Documentation

G.18.2.1 getCPUCoreCount()

`uint32_t BiometricEvaluation::System::getCPUCoreCount ()`

Obtain the number of CPU cores that are online.

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

Returns

The number of CPU cores.

Exceptions

Error::NotImplemented *(p. 636)*	Not implemented for this operating system, or the underlying OS feature is not installed.

G.18.2.2 getCPUCount()

`uint32_t BiometricEvaluation::System::getCPUCount ()`

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

Returns

The number of processing units.

Exceptions

Error::NotImplemented *(p. 636)*	Not implemented for this operating system, or the underlying OS feature is not installed.

G.18.2.3 getCPUSocketCount()

`uint32_t BiometricEvaluation::System::getCPUSocketCount ()`

Obtain the number of CPU sockets that are online.

The hierarchy is CPU (thread) -> Core -> Socket, where there are 1..n hyperthreads per core and 1..n cores per socket.

Returns

The number of CPU sockets.

Exceptions

Error::NotImplemented *(p. 636)*	Not implemented for this operating system, or the underlying OS feature is not installed.

G.18.2.4 getLoadAverage()

`double BiometricEvaluation::System::getLoadAverage ()`

Obtain the system load average for the last minute.

Returns

The system load average.

Exceptions

Error::NotImplemented *(p. 636)*	Not implemented for this operating system, or the underlying OS feature is not installed.

G.18.2.5 getMemInfo()

`std::map< std::string, uint64_t > BiometricEvaluation::System::getMemInfo ()`

Returns

key/value pairs of memory use information

G.18.2.6 getRealMemorySize()

`uint64_t BiometricEvaluation::System::getRealMemorySize ()`

Obtain the amount of real memory in the system.

Returns

The real memory size, in kibibytes.

Exceptions

Error::NotImplemented *(p. 636)*	Not implemented for this operating system, or the underlying OS feature is not installed.

G.19 BiometricEvaluation::Text Namespace Reference

Text (p. 173) processing for string objects.

Functions

- std::string **trimWhitespace** (const std::string &s, const std::locale &locale=std::locale())
  Remove leading and trailing whitespace from a string.
- std::string **ltrimWhitespace** (const std::string &s, const std::locale &locale=std::locale())
  Remove leading whitespace from a string.
- std::string **rtrimWhitespace** (const std::string &s, const std::locale &locale=std::locale())
  Remove trailing whitespace from a string.
- std::string **trim** (const std::string &s, const char trimChar)
  Remove leading and trailing characters from a string.
- std::string **ltrim** (const std::string &s, const char trimChar)
  Remove leading characters from a string.
- std::string **rtrim** (const std::string &s, const char trimChar)
  Remove trailing characters from a string.
- std::string **digest** (const std::string &s, const std::string &digest="md5")
  Compute the digest of a string.
- std::string **digest** (const void *buffer, const size_t buffer_size, const std::string &digest="md5")
  Compute the digest of a memory buffer.
- std::vector< std::string > **split** (const std::string &str, const char delimiter, bool escape=true)
  Return tokens bound by delimiters and the beginning and end of a string.
- std::string **basename** (const std::string &path)
  Extract the filename component of a pathname.
- std::string **dirname** (const std::string &path)

Extract the directory component of a pathname.

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

  Compare two ASCII-encoded strings.
- std::string **toUppercase** (const std::string &str, const std::locale &locale=std::locale())

  Uppercase a string, respecting locale.
- std::string **toLowercase** (const std::string &str, const std::locale &locale=std::locale())

  Lowercase a string, respecting locale.
- std::string **encodeBase64** (const **BiometricEvaluation::Memory::uint8Array** &data)

  Perform Base64 encoding.
- **BiometricEvaluation::Memory::uint8Array decodeBase64** (const std::string &data)

  Perform Base64 decoding.

G.19.1 Detailed Description

Text (p. 173) processing for string objects.

The **Text** (p. 173) package contains a set of functions for the processing of strings: removing leading and trailing whitespace, computing a digest, and other utility functions.

G.19.2 Function Documentation

G.19.2.1 basename()

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

Extract the filename component of a pathname.

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

Parameters

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

Returns

Filename portion of path.

G.19.2.2 caseInsensitiveCompare()

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

Compare two ASCII-encoded strings.

Parameters

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

Returns

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

G.19.2.3 decodeBase64()

```
BiometricEvaluation::Memory::uint8Array BiometricEvaluation::Text::decodeBase64 (
            const std::string & data)
```

Perform Base64 decoding.

Parameters

data	Base64 data to decode.

Returns

Base64 decoding of data.

G.19.2.4 digest() [1/2]

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

Compute the digest of a string.

Parameters

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

Exceptions

Error::MemoryError (p. 604)	Could not allocate memory to store digest.
Error::NotImplemented (p. 636)	The value of digest is not a supported digest.
Error::StrategyError (p. 789)	An error occurred while obtaining the digest.

Returns

An ASCII representation of the hex digits composing the digest.

G.19.2.5 digest() [2/2]

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

Compute the digest of a memory buffer.

Parameters

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

Exceptions

Error::MemoryError (p. 604)	Could not allocate memory to store digest.
Error::NotImplemented (p. 636)	The value of digest is not a supported digest.
Error::StrategyError (p. 789)	An error occurred while obtaining the digest.

Returns

An ASCII representation of the hex digits composing the digest.

G.19.2.6 dirname()

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

Extract the directory component of a pathname.

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

Parameters

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

Returns

Directory portion of path.

G.19.2.7 encodeBase64()

```
std::string BiometricEvaluation::Text::encodeBase64 (
            const  BiometricEvaluation::Memory::uint8Array & data)
```

Perform Base64 encoding.

Parameters

data	Data to encoded.

Returns

Base64 encoding of data.

G.19.2.8 ltrim()

```
std::string BiometricEvaluation::Text::ltrim (
            const std::string & s,
            const char trimChar)
```

Remove leading characters from a string.

Parameters

s	String object whose leading trim↩ Char should be removed.

Parameters

trimChar	Character to remove from the beginning of s.

Returns

Copy of s without leading trimChar.

G.19.2.9 ltrimWhitespace()

```
std::string BiometricEvaluation::Text::ltrimWhitespace (
            const std::string & s,
            const std::locale & locale = std::locale())
```

Remove leading whitespace from a string.

Parameters

s	String object whose leading whitespace should be removed.
locale	Locale to be considered when determining whitespace characters.

Returns

Copy of s without leading whitespace.

G.19.2.10 rtrim()

```
std::string BiometricEvaluation::Text::rtrim (
            const std::string & s,
            const char trimChar)
```

Remove trailing characters from a string.

Parameters

s	String object whose trailing trim↩ Char should be removed.
trimChar	Character to remove from the end of s.

Returns

Copy of s without trailing trimChar.

G.19.2.11 rtrimWhitespace()

```
std::string BiometricEvaluation::Text::rtrimWhitespace (
            const std::string & s,
            const std::locale & locale = std::locale())
```

Remove trailing whitespace from a string.

Parameters

s	String object whose trailing whitespace should be removed.

Parameters

locale	Locale to be considered when determining whitespace characters.

Returns

Copy of s without trailing whitespace.

G.19.2.12 split()

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

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

Parameters

in	*str*	String to tokenize.
in	*delimiter*	Character that defines the end of a token. Any are valid, except '\'.

Parameters

in	*escape*	If the delimiter is prefixed with '\' in the string, do not split at that point and remove the '\'.

Returns

Vector of string tokens, in order of appearance.

Note

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

G.19.2.13 toLowercase()

```
std::string BiometricEvaluation::Text::toLowercase (
            const std::string & str,
            const std::locale & locale = std::locale())
```

Lowercase a string, respecting locale.

Parameters

str	String to lowercase.
locale	Locale to use when lowercasing str.

Returns

Lowercase copy of str.

G.19.2.14 toUppercase()

```
std::string BiometricEvaluation::Text::toUppercase (
            const std::string & str,
            const std::locale & locale = std::locale())
```

Uppercase a string, respecting locale.

Parameters

str	String to uppercase.
locale	Locale to use when uppercasing str.

Returns

Uppercase copy of str.

G.19.2.15 trim()

```
std::string BiometricEvaluation::Text::trim (
            const std::string & s,
            const char trimChar)
```

Remove leading and trailing characters from a string.

Parameters

s	String object whose leading and trailing trim↩ Char should be removed.

Parameters

trimChar	Character to remove from the beginning and ending of s.

Returns

Copy of s without leading or trailing trimChar.

G.19.2.16 trimWhitespace()

```
std::string BiometricEvaluation::Text::trimWhitespace (
            const std::string & s,
            const std::locale & locale = std::locale())
```

Remove leading and trailing whitespace from a string.

Parameters

s	String object whose leading and trailing whitespace should be removed.
locale	Locale to be considered when determining whitespace characters.

Returns

Copy of s without leading or trailing whitespace.

G.20 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** (p. 823) object can be used by applications to limit the amount of processing time taken by a block of code.*

Functions

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

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

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

  *Output stream operator overload for **Timer** (p. 809).*
- void **WatchdogSignalHandler** (int signo, siginfo_t *info, void *uap)

Variables

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

G.20.1 Detailed Description

Support for time and timers.

The **Time** (p. 185) package gathers all timing relating matters, such as Timers, **Watchdog** (p. 823) timers, etc. **Time** (p. 185) values are in microsecond units.

G.20.2 Function Documentation

G.20.2.1 getCurrentCalendarInformation()

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

Obtain customized calendar information.

Parameters

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

Returns

The current calendar information formatted as specified in formatString.

Note

Return value is undefined if format string is invalid.

G.20.2.2 getCurrentDate()

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

Returns

The current ISO 8601 date as a string.

G.20.2.3 getCurrentDateAndTime()

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

Returns

The standard locale current date and time as a string.

G.20.2.4 getCurrentTime()

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

Returns

The current ISO 8601 time as a string.

G.20.2.5 operator<<()

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

Output stream operator overload for **Timer** (p. 809).

Parameters

s	Stream to append.
timer	**Timer** (p. 809) whose elapsed time in microseconds should be appended to s.

Returns

s with value of elapsedStr() appended.

Exceptions

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

G.20.2.6 put_time()

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

Manual implementation of std::put_time.

Note

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

G.21 BiometricEvaluation::Video Namespace Reference

Basic information relating to video and streams.

Classes

- class **Container**

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

Enumerations

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

G.21.1 Detailed Description

Basic information relating to video and streams.

Common representation of a video stream. **Stream** (p. 789) objects can only be obtained from **Container** (p. 374) objects.

The **Video** (p. 187) package gathers all video related matters, including classes to represent a video stream and video containers.

G.21.2 Enumeration Type Documentation

G.21.2.1 CodingFormat

`enum class` **`BiometricEvaluation::Video::CodingFormat`** `[strong]`

Video (p. 187) coding formats.

G.21.2.2 ContainerFormat

`enum class` **`BiometricEvaluation::Video::ContainerFormat`** `[strong]`

Container (p. 374) formats

G.22 BiometricEvaluation::View Namespace Reference

View (p. 819) information.

Classes

- class **AN2KView**

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

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

  A class to represent single biometric element view.

Functions

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

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

  Output stream overload for AN2KQualityMetric.
- std::ostream & **operator**<< (std::ostream &stream, const **AN2KViewVariableResolution::Print**↩**PositionCoordinate** &ppc)

  Output stream overload for PrintPositionCoordinate.

G.22.1 Detailed Description

View (p. 819) information.

The **View** (p. 819) package gathers all classes and other items that are related to a biometric view, which represents an image and all information derived from that image, such as fingerprint minutiae.

G.22.2 Function Documentation

G.22.2.1 operator<<() [1/2]

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

Output stream overload for AN2KQualityMetric.

Parameters

in	*s*	Stream on which to append formatted AN2↩ KQuality↩ Metric information.
in	*qm*	AN2↩ KQuality↩ Metric information to append to stream.

Returns

stream with a qm textual representation appended.

G.22.2.2 operator<<() [2/2]

```
std::ostream & BiometricEvaluation::View::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.

Appendix H

Class Documentation

H.1 _WDIR Struct Reference

Public Attributes

- struct _**wdirent ent**
- WIN32_FIND_DATAW **data**
- int **cached**
- HANDLE **handle**
- wchar_t * **patt**

H.2 _wdirent Struct Reference

Public Attributes

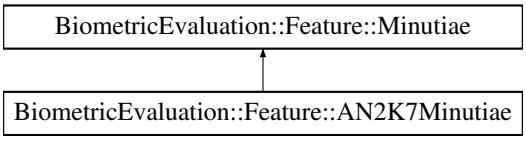
- long **d_ino**
- long **d_off**
- unsigned short **d_reclen**
- size_t **d_namlen**
- int **d_type**
- wchar_t **d_name** [PATH_MAX+1]

H.3 BiometricEvaluation::Feature::AN2K7Minutiae Class Reference

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

`#include <be_feature_an2k7minutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::AN2K7Minutiae:



Classes

- struct **FingerprintReadingSystem**

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

  Pattern classification codes.

Public Types

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

  Methods for encoding minutiae data in an AN2K record.
- using **PatternClassificationSet**
- using **FingerprintReadingSystem**

Public Member Functions

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

  Construct an AN2K7 ***Minutiae*** (p. 615) *object from file data.*
- **AN2K7Minutiae** (**Memory::uint8Array** &buf, int recordNumber)

  Construct an AN2K7 ***Minutiae*** (p. 615) *object from data contained in a memory buffer.*
- PatternClassificationSet **getPatternClassificationSet** () const

  Obtain the set fingerprint pattern classifications.
- **FingerprintReadingSystem** **getOriginatingFingerprintReadingSystem** () const
- Finger::PositionSet **getPositions** () const
- **MinutiaeFormat** **getFormat** () const

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

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

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

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

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

Static Public Member Functions

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

  Convert string read from AN2K record into a ***PatternClassification*** (p. 656).
- static **Finger::PatternClassification** **convertPatternClassification** (const **PatternClassification::↩ Entry** &entry)

  Convert a standard ***PatternClassification::Entry*** (p. 409) *to a PatternClassification::Kind.*
- static **EncodingMethod** **convertEncodingMethod** (const char *mem)

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

  Obtain a Coordinate given an AN2K entry.

H.3.1 Detailed Description

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

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

H.3.2 Member Typedef Documentation

H.3.2.1 FingerprintReadingSystem

```
using BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem
```

Initial value:

```
struct FingerprintReadingSystem
```

H.3.2.2 PatternClassificationSet

```
using BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassificationSet
```

Initial value:

```
std::vector<PatternClassification::Entry>
```

H.3.3 Member Enumeration Documentation

H.3.3.1 EncodingMethod

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

Methods for encoding minutiae data in an AN2K record.

Enumerator

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

H.3.4 Constructor & Destructor Documentation

H.3.4.1 AN2K7Minutiae() [1/2]

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

Construct an AN2K7 **Minutiae** (p. 615) 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 *(p. 420)*	An error occurred when opening or reading from the file.
Error::DataError *(p. 390)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the reques

H.3.4.2 AN2K7Minutiae() [2/2]

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

Construct an AN2K7 **Minutiae** (p. 615) 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 *(p. 390)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the reques

H.3.5 Member Function Documentation

H.3.5.1 convertCoordinate()

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

Obtain a Coordinate given an AN2K entry.

This AN2K entry is formatted as "XXXXYYYY".

Parameters

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

Returns

Image::Coordinate (p. 377) representation of str.

Exceptions

Error::DataError *(p. 390)*	Invalid format of str.

H.3.5.2 convertEncodingMethod()

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

Convert string read from AN2K record into a EncodingMethod.

Parameters

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

Exceptions

Error::DataError *(p. 390)*	Invalid value for mem.

H.3.5.3 convertPatternClassification() [1/2]

```
static Finger::PatternClassification BiometricEvaluation::Feature::AN2K7Minutiae::convert↩
PatternClassification (
            const char * fpc)  [static]
```

Convert string read from AN2K record into a **PatternClassification** (p. 656).

Parameters

in	*fpc*	Value for pattern classification read from AN2K record.

Exceptions

Error::DataError *(p. 390)*	Invalid value for fpc.

H.3.5.4 convertPatternClassification() [2/2]

```
static Finger::PatternClassification BiometricEvaluation::Feature::AN2K7Minutiae::convert↩
PatternClassification (
            const PatternClassification::Entry & entry)  [static]
```

Convert a standard **PatternClassification::Entry** (p. 409) to a PatternClassification::Kind.

Parameters

in	*entry*	A standard pattern classification entry

Exceptions

Error::DataError (p. 390)	Non-standard pattern classification entry.

H.3.5.5 getCores()

`CorePointSet BiometricEvaluation::Feature::AN2K7Minutiae::getCores () const [virtual]`

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

Implements **BiometricEvaluation::Feature::Minutiae** (p. 615).

H.3.5.6 getDeltas()

`DeltaPointSet BiometricEvaluation::Feature::AN2K7Minutiae::getDeltas () const [virtual]`

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

Implements **BiometricEvaluation::Feature::Minutiae** (p. 615).

H.3.5.7 getFormat()

`MinutiaeFormat` `BiometricEvaluation::Feature::AN2K7Minutiae::getFormat () const [virtual]`

Obtain the minutiae format kind.

Implements **BiometricEvaluation::Feature::Minutiae** (p. 616).

H.3.5.8 getMinutiaPoints()

`MinutiaPointSet BiometricEvaluation::Feature::AN2K7Minutiae::getMinutiaPoints () const [virtual]`

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

Implements **BiometricEvaluation::Feature::Minutiae** (p. 616).

H.3.5.9 getOriginatingFingerprintReadingSystem()

`FingerprintReadingSystem` `BiometricEvaluation::Feature::AN2K7Minutiae::getOriginatingFingerprint↩ ReadingSystem () const`

Obtain the originating fingerprint reading system.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The optional OFR field has been excluded.

H.3.5.10 getPatternClassificationSet()

`PatternClassificationSet BiometricEvaluation::Feature::AN2K7Minutiae::getPatternClassification↩ Set () const`

Obtain the set fingerprint pattern classifications.

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

H.3.5.11 getPositions()

`Finger::PositionSet BiometricEvaluation::Feature::AN2K7Minutiae::getPositions () const`

Obtain the set of possible finger positions.

H.3.5.12 getRidgeCountItems()

```
RidgeCountItemSet BiometricEvaluation::Feature::AN2K7Minutiae::getRidgeCountItems () const  [virtual]
```

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

Implements **BiometricEvaluation::Feature::Minutiae** (p. 616).

H.4 BiometricEvaluation::Finger::AN2KMinutiaeDataRecord Class Reference

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

`#include <be_finger_an2kminutiae_data_record.h>`

Public Member Functions

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

  *Construct an **AN2KMinutiaeDataRecord** (p. 199) object from data contained in a file on disk.*
- **AN2KMinutiaeDataRecord** (**Memory::uint8Array** &buf, int recordNumber)

  *Construct an **AN2KMinutiaeDataRecord** (p. 199) object from data contained in a memory buffer.*
- std::shared_ptr< **Feature::AN2K7Minutiae** > **getAN2K7Minutiae** () const

  Obtain the "standard" minutiae data from this Type-9 Record (fields 9.005 - 9.012).
- std::shared_ptr< **Feature::AN2K11EFS::ExtendedFeatureSet** > **getAN2K11EFS** () const

  Obtain the extended feature set data from this Type-9 Record (fields 9.300 - 9.399).
- **Impression** **getImpressionType** () const

  Return impression type field from Type-9 Record.
- std::map< uint16_t, **Memory::uint8Array** > **getRegisteredVendorBlock** (**Feature::Minutiae↩ Format** vendor) const

  Obtain data recorded in a registered vendor minutiae block found in this Type-9 Record.
- int **getIDC** () const

H.4.1 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, registered vendor minutiae data (several vendors from fields 9.013 - 9.175), and extended feature set data (fields 9.300 - 9.399), although not all fields are supported.

H.4.2 Constructor & Destructor Documentation

H.4.2.1 AN2KMinutiaeDataRecord() [1/2]

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

Construct an **AN2KMinutiaeDataRecord** (p. 199) 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 *(p. 420)*	An error occurred when opening or reading from the file.
Error::DataError *(p. 390)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the reques

H.4.2.2 AN2KMinutiaeDataRecord() [2/2]

```
BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::AN2KMinutiaeDataRecord (
            Memory::uint8Array & buf,
          int recordNumber)
```

Construct an **AN2KMinutiaeDataRecord** (p. 199) 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 *(p. 390)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the reques

H.4.3 Member Function Documentation

H.4.3.1 getAN2K11EFS()

`std::shared_ptr< Feature::AN2K11EFS::ExtendedFeatureSet > BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getAN2K11EFS () const`

Obtain the extended feature set data from this Type-9 Record (fields 9.300 - 9.399).

Returns

Shared pointer to an AN2K11ExtendedFeatureSet object if present in the record. The managed pointer will nulptr if there is no extended feature data.

H.4.3.2 getAN2K7Minutiae()

`std::shared_ptr< Feature::AN2K7Minutiae > BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getAN2K7Minutiae () const`

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

Returns

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

H.4.3.3 getIDC()

```
int BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getIDC () const
```

Returns

Information designation character.

H.4.3.4 getImpressionType()

```
Impression BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::getImpressionType () const
```

Return impression type field from Type-9 Record.

Returns

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

H.4.3.5 getRegisteredVendorBlock()

```
std::map< uint16_t, Memory::uint8Array > BiometricEvaluation::Finger::AN2KMinutiaeDataRecord↩
::getRegisteredVendorBlock (
            Feature::MinutiaeFormat vendor) const
```

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

Parameters

in	*vendor*	The vendor whose registered minutiae blocks are being requested.

Returns

A map of the registered vendor minutiae block fields. The map key is the AN2K Field number. The value is a uint8Array of the ASCII data found at that field. All Fields will be present as keys even if there was no data recorded in that Field.

Exceptions

Error::NotImplemented (p. 636)	Cannot return a map of fields for vendor, likely because there exists a better, native implement

H.5 BiometricEvaluation::View::AN2KViewVariableResolution::AN2KQualityMetric Struct Reference

A structure to represent an AN2K quality metric.

`#include <be_view_an2kview_varres.h>`

Public Attributes

- **Feature::FGP fgp**
- uint8_t **score**
- uint16_t **vendorID**
- uint16_t **productCode**

H.5.1 Detailed Description

A structure to represent an AN2K quality metric.

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

H.6 BiometricEvaluation::DataInterchange::AN2KRecord Class Reference

A class to represent an entire ANSI/NIST record.

`#include <be_data_interchange_an2k.h>`

Classes

- struct **CharacterSet**
- struct **DomainName**

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

Public Types

- using **DomainName** = struct DomainName
- using **CharacterSet** = struct CharacterSet

Public Member Functions

- **AN2KRecord** (const std::string filename)

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

  Constructor taking an AN2K record from a buffer.
- std::string **getVersionNumber** () const

- std::string **getDate** () const
- std::string **getDestinationAgency** () const
- std::string **getOriginatingAgency** () const
- std::string **getTransactionControlNumber** () const
- std::string **getNativeScanningResolution** () const
- std::string **getNominalTransmittingResolution** () const
- uint32_t **getFingerLatentCount** () const

  Obtain the count of latent (Type-13) finger views.
- std::vector< **Latent::AN2KView** > **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.
- uint32_t **getFingerFixedResolutionCaptureCount** () const

  Obtain the count of all fixed resolution (Type 3-6) views.
- uint32_t **getFingerFixedResolutionCaptureCount** (const **View::AN2KView::RecordType** type) const

  Obtain the count of fixed resolution (Type 3-6) views.
- std::vector< **Finger::AN2KViewFixedResolution** > **getFingerFixedResolutionCaptures** () const

  Obtain all fixed resolution (Type 3-6) views.
- std::vector< **Finger::AN2KViewFixedResolution** > **getFingerFixedResolutionCaptures** (const **View::AN2KView::RecordType** type) const

  Obtain all fixed resolution (Type 3-6) views of a particular type.
- uint32_t **getPalmCaptureCount** () const

  Obtain the count of capture (Type-15) palm views.
- std::vector< **Palm::AN2KView** > **getPalmCaptures** () const

  Obtain all capture (Type-15) palm views.
- std::vector< **Finger::AN2KMinutiaeDataRecord** > **getMinutiaeDataRecordSet** () const

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

  Obtain the urgency with which a response is required.
- **DomainName getDomainName** () const

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

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

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

Static Public Member Functions

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

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

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

- static bool **isAN2KRecord** (const std::string &filename)

  Determine if file appears to be an ANSI/NIST record.
- static bool **isAN2KRecord** (**BiometricEvaluation::Memory::uint8Array** &buf)

  Determine if file appears to be an ANSI/NIST record.

H.6.1 Detailed Description

A class to represent an entire ANSI/NIST record.

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

H.6.2 Member Typedef Documentation

H.6.2.1 CharacterSet

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

Convenience alias for struct **CharacterSet** (p. 334)

H.6.2.2 DomainName

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

Convenience alias for struct **DomainName** (p. 408)

H.6.3 Constructor & Destructor Documentation

H.6.3.1 AN2KRecord() [1/2]

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

Constructor taking an AN2K record from a file.

Parameters

in	*filename*	The name of the file containing the complete ANSI/NIST record.

Exceptions

Error::FileError (p. 420)	An error occurred when opening or reading the file.
Error::DataError (p. 390)	An error occurred when processing the AN2K record.

H.6.3.2 AN2KRecord() [2/2]

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

Constructor taking an AN2K record from a buffer.

Parameters

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

Exceptions

Error::DataError (p. 390)	An error occurred when processing the AN2K record.

H.6.4 Member Function Documentation

H.6.4.1 getDate()

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

Returns

The date field in the Type-1 record.

H.6.4.2 getDestinationAgency()

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

Returns

The destination agency ID.

H.6.4.3 getDirectoryOfCharacterSets()

```
std::vector< CharacterSet > BiometricEvaluation::DataInterchange::AN2KRecord::getDirectory↩
OfCharacterSets () const
```

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

Returns

Vector of **CharacterSet** (p. 334) structs representing other character sets that may appear in the transaction.

H.6.4.4 getDomainName()

```
DomainName BiometricEvaluation::DataInterchange::AN2KRecord::getDomainName () const
```

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

Returns

DomainName (p. 408) struct with identifier and version information (if defined).

H.6.4.5 getFingerCaptureCount()

```
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.

H.6.4.6 getFingerCaptures()

```
std::vector< Finger::AN2KViewCapture > BiometricEvaluation::DataInterchange::AN2KRecord::get↩
FingerCaptures () const
```

Obtain all capture (Type-14) finger views.

The returned vector will be empty when no capture views are present in the **AN2KRecord** (p. 203).

Returns

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

H.6.4.7 getFingerFixedResolutionCaptureCount() [1/2]

```
uint32_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerFixedResolutionCaptureCount
() const
```

Obtain the count of all fixed resolution (Type 3-6) views.

Returns

The number of fixed resolution captures in the AN2K record.

H.6.4.8 getFingerFixedResolutionCaptureCount() [2/2]

```
uint32_t BiometricEvaluation::DataInterchange::AN2KRecord::getFingerFixedResolutionCaptureCount
(
            const View::AN2KView::RecordType type) const
```

Obtain the count of fixed resolution (Type 3-6) views.

Parameters

type	The fixed resolution record type.

Returns

The number of fixed resolution captures in the AN2K record.

H.6.4.9 getFingerFixedResolutionCaptures() [1/2]

```
std::vector< Finger::AN2KViewFixedResolution > BiometricEvaluation::DataInterchange::AN2KRecord↩
::getFingerFixedResolutionCaptures () const
```

Obtain all fixed resolution (Type 3-6) views.

The returned vector will be empty when no capture views are present in the **AN2KRecord** (p. 203).

Parameters

type	The fixed resolution record type.

Returns

A vector of AN2KViewFixedResolution objects, each representing a single view.

H.6.4.10 getFingerFixedResolutionCaptures() [2/2]

```
std::vector< Finger::AN2KViewFixedResolution > BiometricEvaluation::DataInterchange::AN2KRecord↩
::getFingerFixedResolutionCaptures (
            const View::AN2KView::RecordType type) const
```

Obtain all fixed resolution (Type 3-6) views of a particular type.

The returned vector will be empty when no capture views of the specified type are present in the **AN2↩KRecord** (p. 203).

Parameters

type	The fixed resolution record type.

Returns

Vectors of AN2KViewFixedResolution objects, each representing a single view, separated by type

H.6.4.11 getFingerLatentCount()

```
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.

H.6.4.12 getFingerLatents()

```
std::vector< Latent::AN2KView > BiometricEvaluation::DataInterchange::AN2KRecord::getFinger↩
Latents () const
```

Obtain all latent (Type-13) finger views.

The returned vector will be empty when no latent views are present in the **AN2KRecord** (p. 203).

Returns

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

H.6.4.13 getGreenwichMeanTime()

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

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

Returns

struct tm encoding of the GMT field.

H.6.4.14 getMinutiaeDataRecordSet()

```
std::vector< Finger::AN2KMinutiaeDataRecord > BiometricEvaluation::DataInterchange::AN2KRecord↩
::getMinutiaeDataRecordSet () const
```

Obtain all minutiae (Type-9) data.

Returns

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

H.6.4.15 getNativeScanningResolution()

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

Returns

The native scanning resolution.

H.6.4.16 getNominalTransmittingResolution()

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

Returns

The nominal transmitting resolution.

H.6.4.17 getOriginatingAgency()

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

Returns

The originating agency ID.

H.6.4.18 getPalmCaptureCount()

```
uint32_t BiometricEvaluation::DataInterchange::AN2KRecord::getPalmCaptureCount () const
```

Obtain the count of capture (Type-15) palm views.

Returns

The number of palm captures in the AN2K record.

H.6.4.19 getPalmCaptures()

```
std::vector< Palm::AN2KView > BiometricEvaluation::DataInterchange::AN2KRecord::getPalmCaptures
() const
```

Obtain all capture (Type-15) palm views.

The returned vector will be empty when no capture views are present in the **AN2KRecord** (p. 203).

Returns

A vector of AN2KView objects, each representing a single capture palm view.

H.6.4.20 getPriority()

```
uint8_t BiometricEvaluation::DataInterchange::AN2KRecord::getPriority () const
```

Obtain the urgency with which a response is required.

Returns

Priority (1:High - 9:Low)

H.6.4.21 getTransactionControlNumber()

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

Returns

The transcantion control number.

H.6.4.22 getVersionNumber()

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

Returns

The record version field in the Type-1 record.

H.6.4.23 isAN2KRecord() [1/2]

```
static bool BiometricEvaluation::DataInterchange::AN2KRecord::isAN2KRecord (
            BiometricEvaluation::Memory::uint8Array & buf)  [static]
```

Determine if file appears to be an ANSI/NIST record.

Parameters

buf	**Memory** (p. 156) buffer in question.

Returns

true if the contents of `buf` appears to be an ANSI/NIST record, false otherwise.

H.6.4.24 isAN2KRecord() [2/2]

```
static bool BiometricEvaluation::DataInterchange::AN2KRecord::isAN2KRecord (
            const std::string & filename)  [static]
```

Determine if file appears to be an ANSI/NIST record.

Parameters

filename	Path to a file in question.

Returns

true if the file at `filename` appears to be an ANSI/NIST record, false otherwise.

H.6.4.25 recordLocations() [1/2]

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

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

Parameters

in	*an2k*	ANSI_NIST struct to search.
in	*recordType*	The ID of the Record to search for.

Returns

Set of integer positions within the ANSI_NIST struct where a recordType Record is located.

H.6.4.26 recordLocations() [2/2]

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

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

Parameters

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

Returns

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

Exceptions

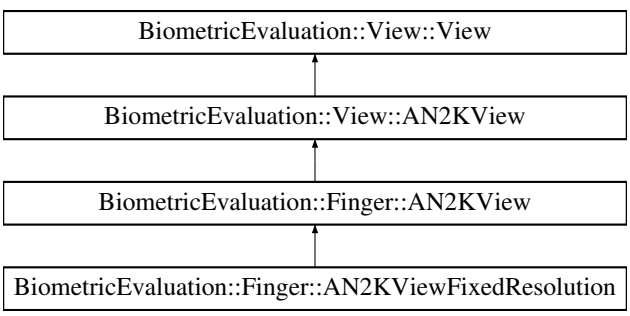
Error::DataError (p. 390)	An error occurred when processing the AN2K record.

H.7 BiometricEvaluation::Finger::AN2KView Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_an2kview.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KView:



Public Member Functions

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

  Obtain the set of minutiae records.
- Finger::PositionSet **getPositions** () const

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

  Obtain the finger impression code.

Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a file.
- **AN2KView** (**Memory::uint8Array** &buf, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a buffer.
- std::vector< **Finger::AN2KMinutiaeDataRecord** > **getMinutiaeDataRecordSet** () const

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

  Obtain the ANSI-NIST record type.
- int **getIDC** () const

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm 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 convertPosition** (int an2kFGP)

  Convert a compression algorithm indicator from an AN2K finger image record.
- static Finger::PositionSet **populateFGP** (FIELD *field)

  Read the finger positions from an AN2K record.
- static **Finger::Impression convertImpression** (const unsigned char *str)

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

Static Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- static **DeviceMonitoringMode convertDeviceMonitoringMode** (const char *dmm)

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

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

Protected Member Functions

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

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

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

  *Add a minutiae data record to the **AN2KMinutiaeDataRecord*** (p. 199) *set.*
- void **setPositions** (Finger::PositionSet &ps)

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

  Mutator for the impression type.

Protected Member Functions inherited from BiometricEvaluation::View::AN2KView

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

  Obtain the complete ANSI/NIST record set.
- RECORD * **getAN2KRecord** () const

  Obtain a pointer to the single ANSI/NIST record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::View::AN2KView

- enum class **RecordType** : uint16_t {
  Type_1 = 1 , **Type_2** = 2 , **Type_3** = 3 , **Type_4** = 4 ,
  Type_5 = 5 , **Type_6** = 6 , **Type_7** = 7 , **Type_8** = 8 ,
  Type_9 = 9 , **Type_10** = 10 , **Type_11** = 11 , **Type_12** = 12 ,
  Type_13 = 13 , **Type_14** = 14 , **Type_15** = 15 , **Type_16** = 16 ,
  Type_17 = 17 , **Type_99** = 99 }
- enum class **DeviceMonitoringMode** {
  Controlled , **Assisted** , **Observed** , **Unattended** ,
  Unknown , **NA** }

  The level of human monitoring for the image capture device.

Static Public Attributes inherited from BiometricEvaluation::View::AN2KView

- 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.

H.7.1 Detailed Description

A class to represent single finger view and derived information.

A base **Finger::AN2KView** (p. 213) 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** (p. 128) object directly.

H.7.2 Constructor & Destructor Documentation

H.7.2.1 AN2KView() [1/2]

```
BiometricEvaluation::Finger::AN2KView::AN2KView (
            const std::string filename,
            const  RecordType typeID,
            const uint32_t recordNumber)  [protected]
```

Construct an AN2K finger view from a file.

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

Parameters

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

Exceptions

Error::ParameterError *(p. 655)*	An invalid parameter was passed in.
Error::DataError *(p. 390)*	An error occurred when parsing the AN2K record.
Error::FileError *(p. 420)*	An error occurred when reading the file.

H.7.2.2 AN2KView() [2/2]

```
BiometricEvaluation::Finger::AN2KView::AN2KView (
                Memory::uint8Array & buf,
              const  RecordType typeID,
              const uint32_t recordNumber)  [protected]
```

Construct an AN2K finger view from a buffer.

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

Parameters

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

Exceptions

Error::ParameterError *(p. 655)*	An invalid parameter was passed in.

Exceptions

Error::DataError *(p. 390)*	An error occurred when parsing the AN2K record.

H.7.3 Member Function Documentation

H.7.3.1 addMinutiaeDataRecord()

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

Add a minutiae data record to the **AN2KMinutiaeDataRecord** (p. 199) set.

Parameters

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

H.7.3.2 convertFingerImageCode()

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

```
@brief
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 *(p. 390)*	The string contains an invalid image code.

H.7.3.3 convertPosition()

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

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

Parameters

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

Exceptions

Error::DataError *(p. 390)*	The position code is invalid.

H.7.3.4 getImpressionType()

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

Obtain the finger impression code.

Returns

The finger impression code.

H.7.3.5 getMinutiaeDataRecordSet()

```
std::vector< AN2KMinutiaeDataRecord > BiometricEvaluation::Finger::AN2KView::getMinutiaeData↩
RecordSet () const
```

Obtain the set of minutiae records.

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

Returns

The vector of minutiae data records.

H.7.3.6 getPositions()

```
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.

H.7.3.7 populateFGP()

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

Read the finger positions from an AN2K record.

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

Exceptions

Error::DataError *(p. 390)*	The data contains an invalid value.

H.7.3.8 setImpressionType()

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

Mutator for the impression type.

Parameters

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

H.7.3.9 setPositions()

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

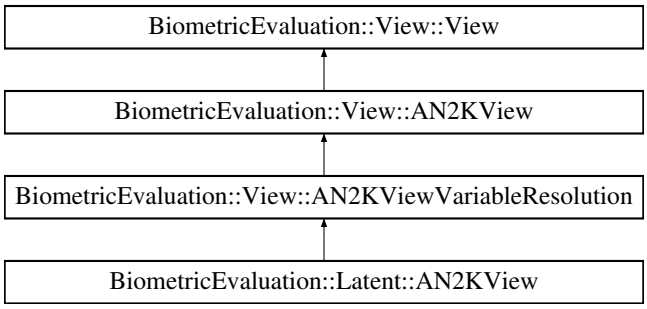
Add a position set to the collection of position sets.

Parameters

in	*ps*	The position set to be added.

H.8 BiometricEvaluation::Latent::AN2KView Class Reference

Inheritance diagram for BiometricEvaluation::Latent::AN2KView:



Public Member Functions

- **AN2KView** (const std::string &filename, const uint32_t recordNumber)

  Construct an AN2K finger view from a file.
- **AN2KView** (**Memory::uint8Array** &buf, const uint32_t recordNumber)

  Construct an AN2K finger view using from a memory buffer.
- Feature::FGPSet **getPositions** () const

  Obtain the set of finger positions.
- QualityMetricSet **getLatentQualityMetric** () const

  Obtain metrics for latent image quality score data for the image stored in this record.
- Finger::PositionDescriptors **getSearchPositionDescriptors** () const

  Return search position descriptors.
- PrintPositionCoordinateSet **getPrintPositionCoordinates** () const

  Obtain print position coordinates.

Public Member Functions inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- **Finger::Impression getImpressionType** () const
- std::string **getSourceAgency** () const
- std::string **getCaptureDate** () const
- std::string **getComment** () const

  Obtain the comment field.
- **Finger::CaptureTechnology getCaptureTechnology** () const

  Obtain capture technology used to create this image.
- **Memory::uint8Array getUserDefinedField** (const uint16_t field) const

  Obtain a user-defined field.

Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a file.
- **AN2KView** (**Memory::uint8Array** &buf, const **RecordType** typeID, const uint32_t recordNumber)

Construct an AN2K view from a buffer.

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

  Obtain the set of minutiae records.

- **RecordType getRecordType** () const

  Obtain the ANSI-NIST record type.

- int **getIDC** () const

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)

- **Image::Size getImageSize** () const

  Obtain the image size.

- **Image::Resolution getImageResolution** () const

  Obtain the image resolution.

- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.

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

  Obtain the compression algorithm used on the image.

- **Image::Resolution getScanResolution** () const

  Obtain the image scan resolution.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- using **AN2KQualityMetric** = struct AN2KQualityMetric
- using **QualityMetricSet** = std::vector< **AN2KQualityMetric**>
- using **PrintPositionCoordinate**
- using **PrintPositionCoordinateSet**

Public Types inherited from BiometricEvaluation::View::AN2KView

- enum class **RecordType** : uint16_t {
  Type_1 = 1 , **Type_2** = 2 , **Type_3** = 3 , **Type_4** = 4 ,
  Type_5 = 5 , **Type_6** = 6 , **Type_7** = 7 , **Type_8** = 8 ,
  Type_9 = 9 , **Type_10** = 10 , **Type_11** = 11 , **Type_12** = 12 ,
  Type_13 = 13 , **Type_14** = 14 , **Type_15** = 15 , **Type_16** = 16 ,
  Type_17 = 17 , **Type_99** = 99 }
- enum class **DeviceMonitoringMode** {
  Controlled , **Assisted** , **Observed** , **Unattended** ,
  Unknown , **NA** }

  The level of human monitoring for the image capture device.

Static Public Member Functions inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- static QualityMetricSet **extractQuality** (FIELD *field, **Feature::PositionType** type)

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

  Read raw bytes from a user-defined AN2K field.
- static **Finger::CaptureTechnology** **convertCaptureTechnology** (const char *str)

  Convert a friction ridge capture technology code from a string.

Static Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- static **DeviceMonitoringMode** **convertDeviceMonitoringMode** (const char *dmm)

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

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

Static Public Attributes inherited from BiometricEvaluation::View::AN2KView

- 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 inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- **AN2KViewVariableResolution** (const std::string &filename, const **RecordType** typeID, const uint32_t recordNumber)

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

  Construct an AN2K finger view using from a memory buffer.
- Feature::FGPSet **getPositions** () const
- Finger::PositionDescriptors **getPositionDescriptors** () const

  Obtain the position descriptors.
- PrintPositionCoordinateSet **getPrintPositionCoordinates** () const

  Obtain print position coordinates.
- QualityMetricSet **getQualityMetric** () const

  Obtain quality metrics for associated image record.

Protected Member Functions inherited from BiometricEvaluation::View::AN2KView

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

  Obtain the complete ANSI/NIST record set.
- RECORD * **getAN2KRecord** () const

  Obtain a pointer to the single ANSI/NIST record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

H.8.1 Constructor & Destructor Documentation

H.8.1.1 AN2KView() [1/2]

```
BiometricEvaluation::Latent::AN2KView::AN2KView (
            const std::string & filename,
            const uint32_t recordNumber)
```

Construct an AN2K finger view from a file.

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

H.8.1.2 AN2KView() [2/2]

```
BiometricEvaluation::Latent::AN2KView::AN2KView (
             Memory::uint8Array & buf,
            const uint32_t recordNumber)
```

Construct an AN2K finger view using from a memory buffer.

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

H.8.2 Member Function Documentation

H.8.2.1 getLatentQualityMetric()

```
QualityMetricSet BiometricEvaluation::Latent::AN2KView::getLatentQualityMetric () const
```

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

Returns

Latent quality metrics

H.8.2.2 getPositions()

`Feature::FGPSet BiometricEvaluation::Latent::AN2KView::getPositions () const`

Obtain the set of finger positions.

An AN2K latent 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.

H.8.2.3 getPrintPositionCoordinates()

`PrintPositionCoordinateSet BiometricEvaluation::Latent::AN2KView::getPrintPositionCoordinates () const`

Obtain print position coordinates.

Returns

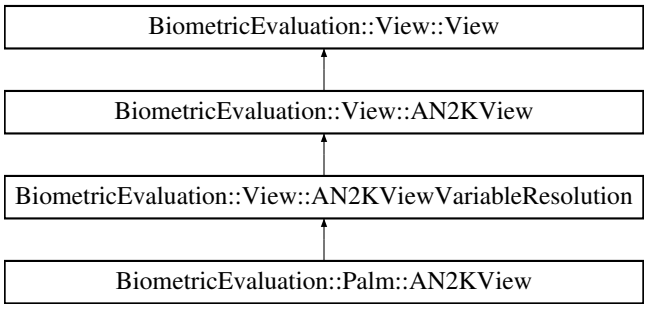
Set of all PrintPositionCoordinates

H.9 BiometricEvaluation::Palm::AN2KView Class Reference

A class to represent a single **Palm** (p. 169) view and derived information.

`#include <be_palm_an2kview.h>`

Inheritance diagram for BiometricEvaluation::Palm::AN2KView:



Public Member Functions

- **AN2KView** (const std::string &filename, const uint32_t recordNumber)

  Construct an AN2K palm view from a file.
- **AN2KView** (**BiometricEvaluation::Memory::uint8Array** &buf, const uint32_t recordNumber)

  Construct an AN2K palm view from a memory buffer.
- **Palm::Position getPosition** () const

  Obtain the palm position.
- QualityMetricSet **getPalmQualityMetric** () const

  Obtain the palm quality metric.

Public Member Functions inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- **Finger::Impression getImpressionType** () const
- std::string **getSourceAgency** () const

- std::string **getCaptureDate** () const
- std::string **getComment** () const

  Obtain the comment field.
- **Finger::CaptureTechnology getCaptureTechnology** () const

  Obtain capture technology used to create this image.
- **Memory::uint8Array getUserDefinedField** (const uint16_t field) const

  Obtain a user-defined field.

Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a file.
- **AN2KView** (**Memory::uint8Array** &buf, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a buffer.
- std::vector< **Finger::AN2KMinutiaeDataRecord** > **getMinutiaeDataRecordSet** () const

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

  Obtain the ANSI-NIST record type.
- int **getIDC** () const

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- using **AN2KQualityMetric** = struct AN2KQualityMetric
- using **QualityMetricSet** = std::vector< **AN2KQualityMetric**>
- using **PrintPositionCoordinate**
- using **PrintPositionCoordinateSet**

Public Types inherited from BiometricEvaluation::View::AN2KView

- enum class **RecordType** : uint16_t {
  Type_1 = 1 , **Type_2** = 2 , **Type_3** = 3 , **Type_4** = 4 ,
  Type_5 = 5 , **Type_6** = 6 , **Type_7** = 7 , **Type_8** = 8 ,
  Type_9 = 9 , **Type_10** = 10 , **Type_11** = 11 , **Type_12** = 12 ,
  Type_13 = 13 , **Type_14** = 14 , **Type_15** = 15 , **Type_16** = 16 ,
  Type_17 = 17 , **Type_99** = 99 }
- enum class **DeviceMonitoringMode** {
  Controlled , **Assisted** , **Observed** , **Unattended** ,
  Unknown , **NA** }

  The level of human monitoring for the image capture device.

Static Public Member Functions inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- static QualityMetricSet **extractQuality** (FIELD *field, **Feature::PositionType** type)

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

  Read raw bytes from a user-defined AN2K field.
- static **Finger::CaptureTechnology convertCaptureTechnology** (const char *str)

  Convert a friction ridge capture technology code from a string.

Static Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- static **DeviceMonitoringMode convertDeviceMonitoringMode** (const char *dmm)

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

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

Static Public Attributes inherited from BiometricEvaluation::View::AN2KView

- 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 inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- **AN2KViewVariableResolution** (const std::string &filename, const **RecordType** typeID, const uint32_t recordNumber)

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

  Construct an AN2K finger view using from a memory buffer.

- Feature::FGPSet **getPositions** () const
- Finger::PositionDescriptors **getPositionDescriptors** () const

  Obtain the position descriptors.
- PrintPositionCoordinateSet **getPrintPositionCoordinates** () const

  Obtain print position coordinates.
- QualityMetricSet **getQualityMetric** () const

  Obtain quality metrics for associated image record.

Protected Member Functions inherited from BiometricEvaluation::View::AN2KView

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

  Obtain the complete ANSI/NIST record set.
- RECORD * **getAN2KRecord** () const

  Obtain a pointer to the single ANSI/NIST record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

H.9.1 Detailed Description

A class to represent a single **Palm** (p. 169) view and derived information.

A **Palm::AN2KView** (p. 226) object represents an ANSI/NIST Type-15 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.

H.9.2 Constructor & Destructor Documentation

H.9.2.1 AN2KView() [1/2]

```
BiometricEvaluation::Palm::AN2KView::AN2KView (
            const std::string & filename,
            const uint32_t recordNumber)
```

Construct an AN2K palm view from a file.

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

H.9.2.2 AN2KView() [2/2]

```
BiometricEvaluation::Palm::AN2KView::AN2KView (
            BiometricEvaluation::Memory::uint8Array & buf,
          const uint32_t recordNumber)
```

Construct an AN2K palm view from a memory buffer.

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

H.9.3 Member Function Documentation

H.9.3.1 getPalmQualityMetric()

```
QualityMetricSet BiometricEvaluation::Palm::AN2KView::getPalmQualityMetric () const
```

Obtain the palm quality metric.

Returns

QualityMetricSet containing the set of metrics the palm image.

H.9.3.2 getPosition()

```
Palm::Position BiometricEvaluation::Palm::AN2KView::getPosition () const
```

Obtain the palm position.

Returns

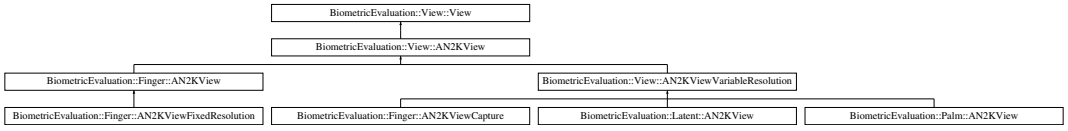
The palm position.

H.10 BiometricEvaluation::View::AN2KView Class Reference

A class to represent single biometric view and derived information.

`#include <be_view_an2kview.h>`

Inheritance diagram for BiometricEvaluation::View::AN2KView:



Public Types

- enum class **RecordType** : uint16_t {
  Type_1 = 1 , **Type_2** = 2 , **Type_3** = 3 , **Type_4** = 4 ,
  Type_5 = 5 , **Type_6** = 6 , **Type_7** = 7 , **Type_8** = 8 ,
  Type_9 = 9 , **Type_10** = 10 , **Type_11** = 11 , **Type_12** = 12 ,
  Type_13 = 13 , **Type_14** = 14 , **Type_15** = 15 , **Type_16** = 16 ,
  Type_17 = 17 , **Type_99** = 99 }
- enum class **DeviceMonitoringMode** {
  Controlled , **Assisted** , **Observed** , **Unattended** ,
  Unknown , **NA** }

  The level of human monitoring for the image capture device.

Public Member Functions

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)
  Construct an AN2K view from a file.
- **AN2KView** (**Memory::uint8Array** &buf, const **RecordType** typeID, const uint32_t recordNumber)
  Construct an AN2K view from a buffer.
- std::vector< **Finger::AN2KMinutiaeDataRecord** > **getMinutiaeDataRecordSet** () const
  Obtain the set of minutiae records.
- **RecordType** **getRecordType** () const
  Obtain the ANSI-NIST record type.
- int **getIDC** () const

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const
  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size** **getImageSize** () const
  Obtain the image size.
- **Image::Resolution** **getImageResolution** () const
  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const
  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm** **getCompressionAlgorithm** () const
  Obtain the compression algorithm used on the image.
- **Image::Resolution** **getScanResolution** () const
  Obtain the image scan resolution.

Static Public Member Functions

- static **DeviceMonitoringMode** **convertDeviceMonitoringMode** (const char *dmm)
  Convert a device monitoring mode indicator from an AN2K record.
- static **Image::CompressionAlgorithm** **convertCompressionAlgorithm** (const uint16_t recordType, const unsigned char *an2kValue)
  Convert a compression algorithm indicator from an AN2K finger image record.

Static Public Attributes

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

Protected Member Functions

- **Memory::AutoBuffer**< ANSI_NIST > **getAN2K** () const
  Obtain the complete ANSI/NIST record set.
- RECORD * **getAN2KRecord** () const
  Obtain a pointer to the single ANSI/NIST record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

H.10.1 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** (p. 128) object directly.

H.10.2 Member Enumeration Documentation

H.10.2.1 DeviceMonitoringMode

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

The level of human monitoring for the image capture device.

Enumerator

Controlled	Operator physically controls the subject to acquire biometric sample.

Enumerator

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

H.10.2.2 RecordType

```
enum class BiometricEvaluation::View::AN2KView::RecordType : uint16_t [strong]
```

The type of AN2K record.

H.10.3 Constructor & Destructor Documentation

H.10.3.1 AN2KView() [1/2]

```
BiometricEvaluation::View::AN2KView::AN2KView (
            const std::string filename,
            const RecordType typeID,
            const uint32_t recordNumber)
```

Construct an AN2K view from a file.

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

H.10.3.2 AN2KView() [2/2]

```
BiometricEvaluation::View::AN2KView::AN2KView (
             Memory::uint8Array & buf,
            const RecordType typeID,
            const uint32_t recordNumber)
```

Construct an AN2K view from a buffer.

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

H.10.4 Member Function Documentation

H.10.4.1 convertCompressionAlgorithm()

```
static Image::CompressionAlgorithm BiometricEvaluation::View::AN2KView::convertCompression↩
Algorithm (
            const uint16_t recordType,
            const unsigned char * an2kValue) [static]
```

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

Parameters

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

Returns

The compression algorithm.

Exceptions

Error::DataError *(p. 390)*	Invalid compression algorithm for record type.
Error::ParameterError *(p. 655)*	Invalid record type.

H.10.4.2 convertDeviceMonitoringMode()

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

Convert a device monitoring mode indicator from an AN2K record.

Parameters

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

Returns

DeviceMonitoringMode representation of dmm.

Exceptions

Error::DataError *(p. 390)*	Invalid format of dmm.

H.10.4.3 getAN2KRecord()

```
RECORD * BiometricEvaluation::View::AN2KView::getAN2KRecord () const  [protected]
```

Obtain a pointer to the single ANSI/NIST record.

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

H.10.4.4 getIDC()

```
int BiometricEvaluation::View::AN2KView::getIDC () const
```

Returns

Information designation character

H.10.4.5 getMinutiaeDataRecordSet()

```
std::vector< Finger::AN2KMinutiaeDataRecord > BiometricEvaluation::View::AN2KView::getMinutiae↩
DataRecordSet () const
```

Obtain the set of minutiae records.

Each **AN2KViewVariableResolution** (p. 250) 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.

H.10.4.6 getRecordType()

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

Obtain the ANSI-NIST record type.

Returns

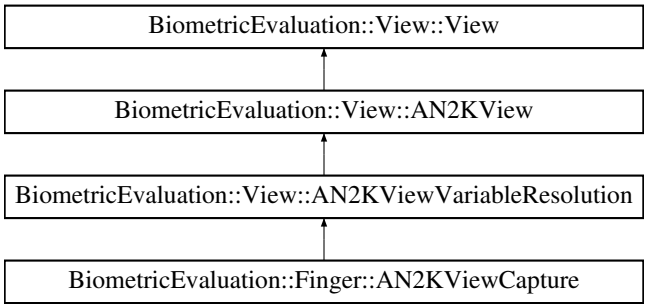
The type of record used to construct this object.

H.11 BiometricEvaluation::Finger::AN2KViewCapture Class Reference

Represents an ANSI/NIST variable-resolution finger image.

`#include <be_finger_an2kview_capture.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewCapture:



Classes

- struct **FingerSegmentPosition**

  Locations of an individual finger segment in a slap.

Public Types

- enum class **AmputatedBandaged** { **Amputated** , **Bandaged** , **NA** }

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

Public Types inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- using **AN2KQualityMetric** = struct AN2KQualityMetric
- using **QualityMetricSet** = std::vector< **AN2KQualityMetric**>
- using **PrintPositionCoordinate**
- using **PrintPositionCoordinateSet**

Public Types inherited from BiometricEvaluation::View::AN2KView

- enum class **RecordType** : uint16_t {
  Type_1 = 1 , **Type_2** = 2 , **Type_3** = 3 , **Type_4** = 4 ,
  Type_5 = 5 , **Type_6** = 6 , **Type_7** = 7 , **Type_8** = 8 ,
  Type_9 = 9 , **Type_10** = 10 , **Type_11** = 11 , **Type_12** = 12 ,
  Type_13 = 13 , **Type_14** = 14 , **Type_15** = 15 , **Type_16** = 16 ,
  Type_17 = 17 , **Type_99** = 99 }
- enum class **DeviceMonitoringMode** {
  Controlled , **Assisted** , **Observed** , **Unattended** ,
  Unknown , **NA** }

  The level of human monitoring for the image capture device.

Public Member Functions

- **AN2KViewCapture** (const std::string &filename, const uint32_t recordNumber)

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

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

  Extract the NQM information from an AN2K FIELD.
- **Finger::Position** **getPosition** () const

  Obtain the finger position.
- PositionDescriptors **getPrintPositionDescriptors** () const

  Return search position descriptors.
- PrintPositionCoordinateSet **getPrintPositionCoordinates** () const

  Obtain print position coordinates.
- QualityMetricSet **getNISTQualityMetric** () const

  Obtain the NIST quality metric for all segmented finger images.
- QualityMetricSet **getSegmentationQualityMetric** () const

  Obtain the segmentation quality metric for all segmented finger images.
- **AmputatedBandaged** **getAmputatedBandaged** () const
- FingerSegmentPositionSet **getFingerSegmentPositionSet** () const
- FingerSegmentPositionSet **getAlternateFingerSegmentPositionSet** () const
- QualityMetricSet **getFingerprintQualityMetric** () const

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

Public Member Functions inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- **Finger::Impression** **getImpressionType** () const
- std::string **getSourceAgency** () const
- std::string **getCaptureDate** () const
- std::string **getComment** () const

  Obtain the comment field.
- **Finger::CaptureTechnology** **getCaptureTechnology** () const

  Obtain capture technology used to create this image.
- **Memory::uint8Array** **getUserDefinedField** (const uint16_t field) const

  Obtain a user-defined field.

Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a file.
- **AN2KView** (**Memory::uint8Array** &buf, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a buffer.
- std::vector< **Finger::AN2KMinutiaeDataRecord** > **getMinutiaeDataRecordSet** () const

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

  Obtain the ANSI-NIST record type.
- int **getIDC** () const

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Additional Inherited Members

Static Public Member Functions inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- static QualityMetricSet **extractQuality** (FIELD *field, **Feature::PositionType** type)

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

  Read raw bytes from a user-defined AN2K field.
- static **Finger::CaptureTechnology convertCaptureTechnology** (const char *str)

  Convert a friction ridge capture technology code from a string.

Static Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- static **DeviceMonitoringMode convertDeviceMonitoringMode** (const char *dmm)

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

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

Static Public Attributes inherited from BiometricEvaluation::View::AN2KView

- 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 inherited from BiometricEvaluation::View::AN2KViewVariableResolution

- **AN2KViewVariableResolution** (const std::string &filename, const **RecordType** typeID, const uint32_t recordNumber)

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

  Construct an AN2K finger view using from a memory buffer.
- Feature::FGPSet **getPositions** () const
- Finger::PositionDescriptors **getPositionDescriptors** () const

  Obtain the position descriptors.
- PrintPositionCoordinateSet **getPrintPositionCoordinates** () const

  Obtain print position coordinates.
- QualityMetricSet **getQualityMetric** () const

  Obtain quality metrics for associated image record.

Protected Member Functions inherited from BiometricEvaluation::View::AN2KView

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

  Obtain the complete ANSI/NIST record set.
- RECORD * **getAN2KRecord** () const

  Obtain a pointer to the single ANSI/NIST record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

H.11.1 Detailed Description

Represents an ANSI/NIST variable-resolution finger image.

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

H.11.2 Member Typedef Documentation

H.11.2.1 FingerSegmentPosition

```
using BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition
```

Initial value:

```
struct FingerSegmentPosition
```

H.11.2.2 FingerSegmentPositionSet

```
using BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPositionSet
```

Initial value:

```
std::vector<FingerSegmentPosition>
```

H.11.3 Member Enumeration Documentation

H.11.3.1 AmputatedBandaged

```
enum class BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged [strong]
```

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

Enumerator

Amputated	Amputation
Bandaged	Unable to print (e.g., bandaged)
NA	Optional field – not specified

H.11.4 Constructor & Destructor Documentation

H.11.4.1 AN2KViewCapture() [1/2]

```
BiometricEvaluation::Finger::AN2KViewCapture::AN2KViewCapture (
            const std::string & filename,
            const uint32_t recordNumber)
```

Construct an AN2K finger view from a file.

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

Parameters

in	*filename*	The name of the file containing the complete ANSI/NIST record.
in	*recordNumber*	The number of variable resolution record to read from the complete AN2K record.

Exceptions

Error::ParameterError (p. 655)	
Error::DataError (p. 390)	
Error::FileError (p. 420)	An error occurred when opening or reading the file.

H.11.4.2 AN2KViewCapture() [2/2]

```
BiometricEvaluation::Finger::AN2KViewCapture::AN2KViewCapture (
                Memory::uint8Array & buf,
                const uint32_t recordNumber)
```

Construct an AN2K finger view using from a memory buffer.

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

H.11.5 Member Function Documentation

H.11.5.1 extractNISTQuality()

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

Extract the NQM information from an AN2K FIELD.

Parameters

field	FIELD containing properly formatted NQM data

Returns

QualityMetricSet representation of field.

Exceptions

Error::DataError *(p. 390)*	Invalid format of field for NQM.

H.11.5.2 getAlternateFingerSegmentPositionSet()

```
FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getAlternateFingerSegment↩
PositionSet () const
```

Returns

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

H.11.5.3 getAmputatedBandaged()

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

Returns

Optional amputated or bandaged code.

H.11.5.4 getFingerprintQualityMetric()

```
QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::getFingerprintQualityMetric ()
const
```

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

Returns

Fingerprint quality metrics

H.11.5.5 getFingerSegmentPositionSet()

```
FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getFingerSegmentPosition↩
Set () const
```

Returns

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

H.11.5.6 getNISTQualityMetric()

```
QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::getNISTQualityMetric () const
```

Obtain the NIST quality metric for all segmented finger images.

Returns

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

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

H.11.5.7 getPosition()

```
Finger::Position BiometricEvaluation::Finger::AN2KViewCapture::getPosition () const
```

Obtain the finger position.

An AN2K finger image record contains a single finger positions. Any minutiae record (Type-9) associated with this image will have its own set of positions.

H.11.5.8 getPrintPositionCoordinates()

```
PrintPositionCoordinateSet BiometricEvaluation::Finger::AN2KViewCapture::getPrintPositionCoordinates
() const
```

Obtain print position coordinates.

Returns

Set of all PrintPositionCoordinates

H.11.5.9 getSegmentationQualityMetric()

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

Obtain the segmentation quality metric for all segmented finger images.

Returns

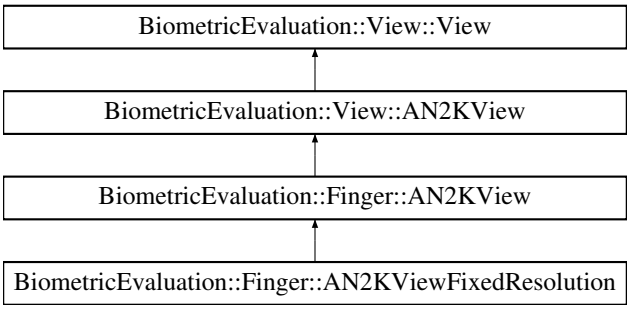
QualityMetricSet containing the segmentation quality metric for all segmented finger images.

H.12 BiometricEvaluation::Finger::AN2KViewFixedResolution Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_an2kview_fixedres.h>`

Inheritance diagram for BiometricEvaluation::Finger::AN2KViewFixedResolution:



Public Member Functions

- **AN2KViewFixedResolution** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

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

  Construct an AN2K finger view from a buffer.

Public Member Functions inherited from BiometricEvaluation::Finger::AN2KView

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

  Obtain the set of minutiae records.
- Finger::PositionSet **getPositions** () const

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

  Obtain the finger impression code.

Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a file.
- **AN2KView** (**Memory::uint8Array** &buf, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a buffer.
- std::vector< **Finger::AN2KMinutiaeDataRecord** > **getMinutiaeDataRecordSet** () const

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

  Obtain the ANSI-NIST record type.
- int **getIDC** () const

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size** **getImageSize** () const

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

Obtain the image resolution.

- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.

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

  Obtain the compression algorithm used on the image.

- **Image::Resolution getScanResolution** () const

  Obtain the image scan resolution.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::View::AN2KView

- enum class **RecordType** : uint16_t {
  Type_1 = 1 , **Type_2** = 2 , **Type_3** = 3 , **Type_4** = 4 ,
  Type_5 = 5 , **Type_6** = 6 , **Type_7** = 7 , **Type_8** = 8 ,
  Type_9 = 9 , **Type_10** = 10 , **Type_11** = 11 , **Type_12** = 12 ,
  Type_13 = 13 , **Type_14** = 14 , **Type_15** = 15 , **Type_16** = 16 ,
  Type_17 = 17 , **Type_99** = 99 }
- enum class **DeviceMonitoringMode** {
  Controlled , **Assisted** , **Observed** , **Unattended** ,
  Unknown , **NA** }

  The level of human monitoring for the image capture device.

Static Public Member Functions inherited from BiometricEvaluation::Finger::AN2KView

- static **Finger::Position convertPosition** (int an2kFGP)

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

- static Finger::PositionSet **populateFGP** (FIELD *field)

  Read the finger positions from an AN2K record.

- static **Finger::Impression convertImpression** (const unsigned char *str)

  Convert an impression code from a string.

- static **Finger::FingerImageCode convertFingerImageCode** (const char *str)

Static Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- static **DeviceMonitoringMode convertDeviceMonitoringMode** (const char *dmm)

  Convert a device monitoring mode indicator from an AN2K record.

- static **Image::CompressionAlgorithm convertCompressionAlgorithm** (const uint16_t recordType, const unsigned char *an2kValue)

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

Static Public Attributes inherited from BiometricEvaluation::View::AN2KView

- 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 inherited from BiometricEvaluation::Finger::AN2KView

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

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

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

  *Add a minutiae data record to the **AN2KMinutiaeDataRecord*** (p. *199*) *set.*
- void **setPositions** (Finger::PositionSet &ps)

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

  Mutator for the impression type.

Protected Member Functions inherited from BiometricEvaluation::View::AN2KView

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

  Obtain the complete ANSI/NIST record set.
- RECORD * **getAN2KRecord** () const

  Obtain a pointer to the single ANSI/NIST record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

H.12.1 Detailed Description

A class to represent single finger view and derived information.

A base **Finger::AN2KView** (p. 213) 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** (p. 128) object directly.

H.12.2 Constructor & Destructor Documentation

H.12.2.1 AN2KViewFixedResolution() [1/2]

```
BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution (
            const std::string filename,
            const  RecordType typeID,
            const uint32_t recordNumber)
```

Construct an AN2K finger view from a file.

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

Parameters

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

Exceptions

Error::ParameterError (p. 655)	An invalid parameter was passed in.
Error::DataError (p. 390)	An error occurred when parsing the AN2K record.
Error::FileError (p. 420)	An error occurred when reading the file.

H.12.2.2 AN2KViewFixedResolution() [2/2]

```
BiometricEvaluation::Finger::AN2KViewFixedResolution::AN2KViewFixedResolution (
            Memory::uint8Array & buf,
          const  RecordType typeID,
          const uint32_t recordNumber)
```

Construct an AN2K finger view from a buffer.

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

Parameters

in	*buf*	The buffer containing the AN2K record.
in	*typeID*	The type of AN2K finger view: Type-3/↩ Type-4/etc.

Parameters

in	*recordNumber*	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

Exceptions

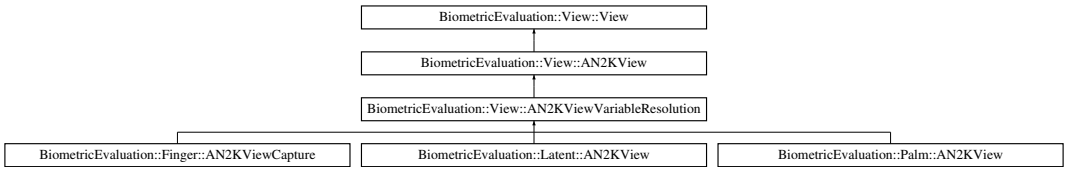
Error::ParameterError *(p. 655)*	An invalid parameter was passed in.
Error::DataError *(p. 390)*	An error occurred when parsing the AN2K record.

H.13 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.
- struct **PrintPositionCoordinate**

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

Public Types

- using **AN2KQualityMetric** = struct AN2KQualityMetric
- using **QualityMetricSet** = std::vector< **AN2KQualityMetric** >
- using **PrintPositionCoordinate**
- using **PrintPositionCoordinateSet**

Public Types inherited from BiometricEvaluation::View::AN2KView

- enum class **RecordType** : uint16_t {
  Type_1 = 1 , **Type_2** = 2 , **Type_3** = 3 , **Type_4** = 4 ,
  Type_5 = 5 , **Type_6** = 6 , **Type_7** = 7 , **Type_8** = 8 ,
  Type_9 = 9 , **Type_10** = 10 , **Type_11** = 11 , **Type_12** = 12 ,
  Type_13 = 13 , **Type_14** = 14 , **Type_15** = 15 , **Type_16** = 16 ,
  Type_17 = 17 , **Type_99** = 99 }
- enum class **DeviceMonitoringMode** {
  Controlled , **Assisted** , **Observed** , **Unattended** ,
  Unknown , **NA** }

  The level of human monitoring for the image capture device.

Public Member Functions

- **Finger::Impression getImpressionType** () const
- std::string **getSourceAgency** () const
- std::string **getCaptureDate** () const
- std::string **getComment** () const

  Obtain the comment field.
- **Finger::CaptureTechnology getCaptureTechnology** () const

  Obtain capture technology used to create this image.
- **Memory::uint8Array getUserDefinedField** (const uint16_t field) const

  Obtain a user-defined field.

Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a file.
- **AN2KView** (**Memory::uint8Array** &buf, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a buffer.
- std::vector< **Finger::AN2KMinutiaeDataRecord** > **getMinutiaeDataRecordSet** () const

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

  Obtain the ANSI-NIST record type.
- int **getIDC** () const

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size** **getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm** **getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Static Public Member Functions

- static QualityMetricSet **extractQuality** (FIELD *field, **Feature::PositionType** type)

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

  Read raw bytes from a user-defined AN2K field.
- static **Finger::CaptureTechnology** **convertCaptureTechnology** (const char *str)

  Convert a friction ridge capture technology code from a string.

Static Public Member Functions inherited from BiometricEvaluation::View::AN2KView

- static **DeviceMonitoringMode** **convertDeviceMonitoringMode** (const char *dmm)

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

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

Protected Member Functions

- **AN2KViewVariableResolution** (const std::string &filename, const **RecordType** typeID, const uint32_t recordNumber)

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

  Construct an AN2K finger view using from a memory buffer.
- Feature::FGPSet **getPositions** () const
- Finger::PositionDescriptors **getPositionDescriptors** () const

  Obtain the position descriptors.
- PrintPositionCoordinateSet **getPrintPositionCoordinates** () const

  Obtain print position coordinates.
- QualityMetricSet **getQualityMetric** () const

  Obtain quality metrics for associated image record.

Protected Member Functions inherited from BiometricEvaluation::View::AN2KView

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

  Obtain the complete ANSI/NIST record set.
- RECORD * **getAN2KRecord** () const

  Obtain a pointer to the single ANSI/NIST record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Additional Inherited Members

Static Public Attributes inherited from BiometricEvaluation::View::AN2KView

- 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.

H.13.1 Detailed Description

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

The view represents a variable resolution (Type-13/14/15) AN2K record.

H.13.2 Member Typedef Documentation

H.13.2.1 PrintPositionCoordinate

```
using BiometricEvaluation::View::AN2KViewVariableResolution::PrintPositionCoordinate
```

Initial value:

```
struct PrintPositionCoordinate
```

H.13.2.2 PrintPositionCoordinateSet

```
using BiometricEvaluation::View::AN2KViewVariableResolution::PrintPositionCoordinateSet
```

Initial value:

```
std::vector<PrintPositionCoordinate>
```

H.13.3 Constructor & Destructor Documentation

H.13.3.1 AN2KViewVariableResolution() [1/2]

```
BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution (
            const std::string & filename,
            const  RecordType typeID,
            const uint32_t recordNumber)  [protected]
```

Construct an AN2K finger view from a file.

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

H.13.3.2 AN2KViewVariableResolution() [2/2]

```
BiometricEvaluation::View::AN2KViewVariableResolution::AN2KViewVariableResolution (
             Memory::uint8Array & buf,
            const  RecordType typeID,
            const uint32_t recordNumber)  [protected]
```

Construct an AN2K finger view using from a memory buffer.

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

H.13.4 Member Function Documentation

H.13.4.1 convertCaptureTechnology()

```
static  Finger::CaptureTechnology BiometricEvaluation::View::AN2KViewVariableResolution::convert↩
CaptureTechnology (
            const char * str)  [static]
```

Convert a friction ridge capture technology code from a string.

Parameters

str	String read from ANSI/↩ NIST file repre-sent-ing a FRCT code.

Returns

Decoded CaptureTechnology.

Exceptions

Error::ObjectDoesNotExist (p. 637)	Invalid FRCT code encoded within str.

H.13.4.2 extractQuality()

```
static QualityMetricSet BiometricEvaluation::View::AN2KViewVariableResolution::extractQuality
(
            FIELD * field,
             Feature::PositionType type)  [static]
```

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

Parameters

in	*field*	A pointer to the field within the AN2K record.
in	*type*	The position type.

Exceptions

Error::DataError *(p. 390)*	The data contains an invalid value.

H.13.4.3 getCaptureDate()

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

Returns

The capture date.

H.13.4.4 getCaptureTechnology()

```
 Finger::CaptureTechnology BiometricEvaluation::View::AN2KViewVariableResolution::getCapture↩
Technology () const
```

Obtain capture technology used to create this image.

Returns

Capture technology used to create this image.

H.13.4.5 getComment()

```
std::string BiometricEvaluation::View::AN2KViewVariableResolution::getComment () const
```

Obtain the comment field.

The comment field is optional in an AN2K record.

Returns

The comment field, empty string if not present.

H.13.4.6 getImpressionType()

```
Finger::Impression BiometricEvaluation::View::AN2KViewVariableResolution::getImpressionType
() const
```

Returns

The finger/palm impression code.

H.13.4.7 getPositionDescriptors()

```
Finger::PositionDescriptors BiometricEvaluation::View::AN2KViewVariableResolution::getPosition↩
Descriptors () const  [protected]
```

Obtain the position descriptors.

Subclasses specialize the position descriptors based on the semantic meaning pertinent for that class.

Returns

The set of position descriptors.

H.13.4.8 getPositions()

```
Feature::FGPSet BiometricEvaluation::View::AN2KViewVariableResolution::getPositions () const
[protected]
```

```
        @brief
        Obtain the set of finger positions.
        @details
        An AN2K variable resolution image record may contain
```

a set of possible friction ridge positions. This method returns that set as read from the image record. Subclasses must retrieve the position information relevant to that class.

Returns

The set of friction ridge generalized positions.

H.13.4.9 getPrintPositionCoordinates()

```
PrintPositionCoordinateSet BiometricEvaluation::View::AN2KViewVariableResolution::getPrint↩
PositionCoordinates () const  [protected]
```

Obtain print position coordinates.

Returns

Set of all PrintPositionCoordinates

H.13.4.10 getQualityMetric()

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

Obtain quality metrics for associated image record.

Returns

Quality metrics

H.13.4.11 getSourceAgency()

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

Returns

The source agency.

H.13.4.12 getUserDefinedField()

```
Memory::uint8Array BiometricEvaluation::View::AN2KViewVariableResolution::getUserDefinedField
(
            const uint16_t field) const
```

Obtain a user-defined field.

Fields are retrieved on-demand and then cached.

Parameters

in	*field*	The field number to retrieve.

Returns

Raw bytes read from the field.

Exceptions

Error::ObjectDoesNotExist (p. 637)	There is no user-defined field with the requested field number.
Error::ParameterError (p. 655)	Invalid value for field.
Error::StrategyError (p. 789)	Field could not be cached.

H.13.4.13 parseUserDefinedField()

```
static Memory::uint8Array BiometricEvaluation::View::AN2KViewVariableResolution::parseUser↩
DefinedField (
            const RECORD *const record,
            int fieldID) [static]
```

Read raw bytes from a user-defined AN2K field.

Parameters

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

Returns

Raw bytes from field.

Exceptions

Error::ObjectDoesNotExist (p. 637)	There is no user-defined field with the requested field number.
Error::ParameterError (p. 655)	Invalid value for fieldID.

H.14 BiometricEvaluation::Feature::Sort::Angle Class Reference

`#include <be_feature_sort.h>`

Public Member Functions

- bool **operator()** (const **BiometricEvaluation::Feature::MinutiaPoint** &lhs, const **BiometricEvaluation::Feature::MinutiaPoint** &rhs) const

H.14.1 Detailed Description

Sort (p. 117) by increasing angle (theta)

H.14.2 Member Function Documentation

H.14.2.1 operator()()

```
bool BiometricEvaluation::Feature::Sort::Angle::operator() (
            const  BiometricEvaluation::Feature::MinutiaPoint & lhs,
            const  BiometricEvaluation::Feature::MinutiaPoint & rhs) const
```

MinutiaPoint (p. 619) angle ascending comparator.

H.15 BiometricEvaluation::DataInterchange::ANSI2004Record Class Reference

`#include <be_data_interchange_ansi2004.h>`

Public Member Functions

- **ANSI2004Record** (const **BiometricEvaluation::Memory::uint8Array** &fmr, const **BiometricEvaluation::Memory::uint8Array** &fir)

  ANSI2004Record *(p. 259) constructor using a pair of finger minutia and image records.*
- **ANSI2004Record** (const std::string &fmrPath, const std::string &firPath)

  ANSI2004Record *(p. 259) constructor using a pair of finger minutia and image records.*
- **ANSI2004Record** (const std::initializer_list< **BiometricEvaluation::Finger::ANSI2004View** > &views)

  ANSI2004Record *(p. 259) constructor using a set of finger view records.*
- **Finger::ANSI2004View getView** (const uint64_t viewNumber) const

  Obtain an ANSI2004View.
- uint64_t **insertView** (const **Finger::ANSI2004View** &view)

  Insert a finger view to the record at a specific position.
- uint64_t **insertView** (const **Finger::ANSI2004View** &view, const uint64_t viewNumber)

  Insert a finger view to the record at a specific position.
- uint64_t **updateView** (const **Finger::ANSI2004View** &view, const uint64_t viewNumber)

  Update an entire finger view.
- void **removeView** (const uint64_t viewNumber)

  Remove a view from the record.
- void **isolateView** (const uint64_t viewNumber)

  Isolate a finger view from the record.
- std::vector< **BiometricEvaluation::Feature::INCITSMinutiae** > **getMinutia** () const

  Obtain the INCITSMinutiae for all finger views.
- **BiometricEvaluation::Feature::INCITSMinutiae getMinutia** (uint32_t viewNumber) const

  Obtain the INCITSMinutiae for a finger view.
- void **setMinutia** (const std::vector< **BiometricEvaluation::Feature::INCITSMinutiae** > &minutia)

  Alter the minutia for every finger view.
- void **setMinutia** (uint32_t viewNumber, const **BiometricEvaluation::Feature::INCITSMinutiae** &minutia)

  Alter the minutia for a single finger view.
- **BiometricEvaluation::Memory::uint8Array getFMR** () const

  Obtain an ANSI/INCITS 378-2004 record.
- uint64_t **getNumFingerViews** () const

  Obtain the number of finger views in this finger minutia record.

Protected Member Functions

- uint64_t **getFMRLength** () const

  Obtain the size of FMR that will be written by ***getFMR()*** *(p. 261).*
- uint64_t **getEDBLength** () const

  Obtain the size of EDB that will be written by ***getFMR()*** *(p. 261).*

H.15.1 Detailed Description

All finger views from a single finger minutiae record

H.15.2 Constructor & Destructor Documentation

H.15.2.1 ANSI2004Record() [1/3]

```
BiometricEvaluation::DataInterchange::ANSI2004Record::ANSI2004Record (
            const  BiometricEvaluation::Memory::uint8Array & fmr,
            const  BiometricEvaluation::Memory::uint8Array & fir)
```

ANSI2004Record (p. 259) constructor using a pair of finger minutia and image records.

One or both records can be the empty array. The data obtained from an empty record will be set to the zero-value.

Parameters

fmr	**Finger** (p. 122) minutia record.
fir	**Finger** (p. 122) image record.

H.15.2.2 ANSI2004Record() [2/3]

```
BiometricEvaluation::DataInterchange::ANSI2004Record::ANSI2004Record (
            const std::string & fmrPath,
            const std::string & firPath)
```

ANSI2004Record (p. 259) constructor using a pair of finger minutia and image records.

One or both records can be the empty string. The data obtained from an empty record will be set to the zero-value.

Parameters

fmr	Path to a finger minutia record.
fir	Path to a finger image record.

H.15.2.3 ANSI2004Record() [3/3]

```
BiometricEvaluation::DataInterchange::ANSI2004Record::ANSI2004Record (
            const std::initializer_list< BiometricEvaluation::Finger::ANSI2004View > & views)
```

ANSI2004Record (p. 259) constructor using a set of finger view records.

Parameters

views	ANSI2004↩ View objects.

H.15.3 Member Function Documentation

H.15.3.1 getEDBLength()

```
uint64_t BiometricEvaluation::DataInterchange::ANSI2004Record::getEDBLength () const [protected]
```

Obtain the size of EDB that will be written by **getFMR()** (p. 261).

Even if unmodified after reading a record, this value may be different than expected because ANSI2004↩ View does not support reading proprietary extended data blocks.

Returns

Size of EDB that will be returned from **getFMR()** (p. 261).

@seealso **getFMR()** (p. 261)

H.15.3.2 getFMR()

```
BiometricEvaluation::Memory::uint8Array BiometricEvaluation::DataInterchange::ANSI2004Record↩
::getFMR () const
```

Obtain an ANSI/INCITS 378-2004 record.

Note

Reflects the current state of the object contained within.

Returns

A well-formed ANSI/INCITS 378-2004 record.

H.15.3.3 getFMRLength()

```
uint64_t BiometricEvaluation::DataInterchange::ANSI2004Record::getFMRLength () const [protected]
```

Obtain the size of FMR that will be written by **getFMR()** (p. 261).

Even if unmodified after reading a record, this value may be different than expected because ANSI2004↩ View does not support reading proprietary extended data blocks.

Returns

Size of FMR that will be returned from **getFMR()** (p. 261).

@seealso **getFMR()** (p. 261) @seealso **getEDBLength()** (p. 261)

H.15.3.4 getMinutia() [1/2]

std::vector< **BiometricEvaluation::Feature::INCITSMinutiae** > BiometricEvaluation::DataInterchange↩
::ANSI2004Record::getMinutia () const

Obtain the INCITSMinutiae for all finger views.

Returns

Vector of INCITSMinutiae for all finger views in this record.

H.15.3.5 getMinutia() [2/2]

BiometricEvaluation::Feature::INCITSMinutiae BiometricEvaluation::DataInterchange::ANSI2004↩
Record::getMinutia (
uint32_t *viewNumber*) const

Obtain the INCITSMinutiae for a finger view.

Parameters

viewNumber	1-based finger view whose minutia will be returned.

Returns

INCITSMinutiae for finger view viewNumber.

H.15.3.6 getNumFingerViews()

uint64_t BiometricEvaluation::DataInterchange::ANSI2004Record::getNumFingerViews () const

Obtain the number of finger views in this finger minutia record.

Returns

Number of finger views, as iterated over when constructing this object.

H.15.3.7 getView()

Finger::ANSI2004View BiometricEvaluation::DataInterchange::ANSI2004Record::getView (
const uint64_t *viewNumber*) const

Obtain an ANSI2004View.

Parameters

viewNumber	The position of the view to obtain.

Returns

ANSI2004View for view number viewNumber.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	viewNumber does not exist.

H.15.3.8 insertView() [1/2]

```
uint64_t BiometricEvaluation::DataInterchange::ANSI2004Record::insertView (
            const  Finger::ANSI2004View & view)
```

Insert a finger view to the record at a specific position.

Parameters

view	**Finger** (p. 122) view to add.

Returns

View (p. 188) number for view in this record.

H.15.3.9 insertView() [2/2]

```
uint64_t BiometricEvaluation::DataInterchange::ANSI2004Record::insertView (
            const  Finger::ANSI2004View & view,
            const uint64_t viewNumber)
```

Insert a finger view to the record at a specific position.

Parameters

view	**Finger** (p. 122) view to add.

Parameters

viewNumber	**View** (p. 188) number to assign to this view.

Returns

The view number.

Exceptions

BiometricEvaluation::Error::StrategyError *(p. 789)*	viewNumber is not valid.

H.15.3.10 isolateView()

```
void BiometricEvaluation::DataInterchange::ANSI2004Record::isolateView (
            const uint64_t viewNumber)
```

Isolate a finger view from the record.

Parameters

viewNumber	The view number to isolate.

Exceptions

BiometricEvaluation::Error::ObjectDoesNotExist *(p. 637)*	viewNumber does not exist.

Note

The remaining view becomes view 1.

H.15.3.11 removeView()

```
void BiometricEvaluation::DataInterchange::ANSI2004Record::removeView (
            const uint64_t viewNumber)
```

Remove a view from the record.

Parameters

viewNumber	The view number to remove.

Exceptions

BiometricEvaluation::Error::ObjectDoesNotExist (p. 637)	viewNumber does not exist.

Note

All views will be renumbered after removal.

H.15.3.12 setMinutia() [1/2]

```
void BiometricEvaluation::DataInterchange::ANSI2004Record::setMinutia (
            const std::vector< BiometricEvaluation::Feature::INCITSMinutiae > & minutia)
```

Alter the minutia for every finger view.

Parameters

minutia	A vector of INCITSMinutiae for each finger view.

Exceptions

Error::StrategyError (p. 789)	Size of minutia does not equal the number of finger views in this record.

H.15.3.13 setMinutia() [2/2]

```
void BiometricEvaluation::DataInterchange::ANSI2004Record::setMinutia (
            uint32_t viewNumber,
            const BiometricEvaluation::Feature::INCITSMinutiae & minutia)
```

Alter the minutia for a single finger view.

Parameters

viewNumber	1-based finger view whose minutia will be replaced.
minutia	INCITSMinutiae for finger view viewNumber.

Exceptions

Error::StrategyError *(p. 789)*	**View** (p. 188) number is invalid for this finger record.

H.15.3.14 updateView()

```
uint64_t BiometricEvaluation::DataInterchange::ANSI2004Record::updateView (
            const  Finger::ANSI2004View & view,
            const uint64_t viewNumber)
```

Update an entire finger view.

Parameters

view	Updated finger view.
viewNumber	**View** (p. 188) number replaced by view.

Returns

The view number.

Exceptions

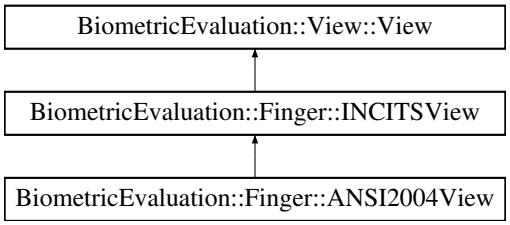
BiometricEvaluation::Error::StrategyError *(p. 789)*	viewNumber is not valid.

H.16 BiometricEvaluation::Finger::ANSI2004View Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_ansi2004view.h>`

Inheritance diagram for BiometricEvaluation::Finger::ANSI2004View:



Public Member Functions

- **ANSI2004View** ()

  Construct an empty ANSI finger view.
- **ANSI2004View** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber)

  Construct an ANSI-2004 finger view from records contained in files.
- **ANSI2004View** (const **Memory::uint8Array** &fmrBuffer, const **Memory::uint8Array** &firBuffer, const uint32_t viewNumber)

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

Public Member Functions inherited from BiometricEvaluation::Finger::INCITSView

- **Feature::INCITSMinutiae getMinutiaeData** () const

  Obtain the set of minutiae records.
- **Finger::Position getPosition** () const

  Obtain the finger position.
- **Finger::Impression getImpressionType** () const

  Obtain the finger impression code.
- uint32_t **getQuality** () const

  Obtain the finger quality value.
- uint16_t **getCaptureEquipmentID** () const

  Obtain the capture equipment identifier.
- bool **isAppendixFCompliant** () const

  Obtain the capture equipment compliance indicator for 'Appendix F'.
- uint16_t **getProductIDOwner** () const

  Obtain the CBEFF product identifier owner.

- uint16_t **getProductIDType** () const

  Obtain the CBEFF product identifier type.
- uint32_t **getRecordLength** () const
- uint8_t **getNumFingerViews** () const
- uint8_t **getFMRReservedByte** () const
- uint32_t **getViewNumber** () const
- uint16_t **getEDBLength** () const
- std::vector< uint8_t > **getMinutiaeReservedData** () const
- void **setMinutiaeData** (const **Feature::INCITSMinutiae** &fmd)

  *Mutator for the **Feature::INCITSMinutiae*** (p. 495) *item.*
- void **setMinutiaeReservedData** (const std::vector< uint8_t > &reservedBits)

  Mutator for the FMD reserved bits vector.

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size** **getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm** **getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Protected Member Functions

- void **readFMRHeader** (**Memory::IndexedBuffer** &buf)
- void **readCoreDeltaData** (**Memory::IndexedBuffer** &buf, uint32_t dataLength, Feature::CorePoint↩ Set &cores, Feature::DeltaPointSet &deltas)

  Read the core points data.

Protected Member Functions inherited from BiometricEvaluation::Finger::INCITSView

- **INCITSView** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view↩ Number)

  Construct the common components of an INCITS finger view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &fmrBuffer, const **Memory::uint8Array** &firBuffer, const uint32_t viewNumber)

  Construct an INCITS finger view from records contained in buffers.
- **Memory::uint8Array** const & **getFMRData** () const

  Obtain a reference to the finger minutiae record data buffer.
- **Memory::uint8Array** const & **getFIRData** () const

Obtain a reference to the finger image record data buffer.

- void **setPosition** (const **Finger::Position** &position)

  Mutator for the position.
- void **setImpressionType** (const **Finger::Impression** &impression)

  Mutator for the impression type.
- void **setQuality** (uint32_t quality)

  Mutator for the finger quality value.
- void **setViewNumber** (uint32_t viewNumber)

  Mutator for the finger view number.
- void **setCaptureEquipmentID** (uint16_t id)

  Mutator for the equipment ID.
- void **setCBEFFProductIDs** (uint16_t owner, uint16_t type)

  Mutator for the CBEFF Product ID owner and type.
- void **setAppendixFCompliance** (bool flag)

  Mutator for the Appendix F compliance indicator.
- void **readFMRHeader** (**Memory::IndexedBuffer** &buf, const uint32_t formatStandard)

  Read the common finger minutiae record header from an INCITS record.
- void **readFVMR** (**Memory::IndexedBuffer** &buf)

  Read the common finger view record information from an INCITS record.
- virtual std::tuple< Feature::MinutiaPointSet, std::vector< uint8_t > > **readMinutiaeDataPoints** (**Memory::IndexedBuffer** &buf, uint32_t count)

  Read the minutiae data points, and extended data blocks.
- virtual void **readExtendedDataBlock** (**Memory::IndexedBuffer** &buf)

  Read the common extended data block.
- virtual Feature::RidgeCountItemSet **readRidgeCountData** (**Memory::IndexedBuffer** &buf, uint32_t dataLength)

  Read the ridge count data.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x20323000

Static Protected Attributes inherited from BiometricEvaluation::Finger::INCITSView

- static const uint32_t **FMR_BASE_FORMAT_ID** = 0x464D5200
- static const uint32_t **ANSI2004_STANDARD** = 1

  The type of record that will be read by the subclass.
- static const uint32_t **ISO2005_STANDARD** = 2
- static const uint32_t **ANSI2007_STANDARD** = 3

Additional Inherited Members

Static Public Member Functions inherited from BiometricEvaluation::Finger::INCITSView

- static **Finger::Position convertPosition** (int incitsFGP)

  Convert a finger postion code from an INCITS finger record to the common code.
- static **Finger::Impression convertImpression** (int incitsIMP)

  Convert a impression type code from an INCITS finger record to the common code.

H.16.1 Detailed Description

A class to represent single finger view and derived information.

A **Finger::ANSI2004View** (p. 267) object represents a finger view from a INCITS/ANSI-2004 **Finger** (p. 122) Minutiae Record.

H.16.2 Constructor & Destructor Documentation

H.16.2.1 ANSI2004View() [1/2]

```
BiometricEvaluation::Finger::ANSI2004View::ANSI2004View (
            const std::string & fmrFilename,
            const std::string & firFilename,
            const uint32_t viewNumber)
```

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

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

Parameters

in	*fmrFilename*	The name of the file containing the complete finger minutiae record.

Parameters

in	*firFilename*	The name of the file containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

H.16.2.2 ANSI2004View() [2/2]

```
BiometricEvaluation::Finger::ANSI2004View::ANSI2004View (
            const  Memory::uint8Array & fmrBuffer,
            const  Memory::uint8Array & firBuffer,
            const uint32_t viewNumber)
```

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

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

Parameters

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

Parameters

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

H.16.3 Member Function Documentation

H.16.3.1 readCoreDeltaData()

```
void BiometricEvaluation::Finger::ANSI2004View::readCoreDeltaData (
             Memory::IndexedBuffer & buf,
            uint32_t dataLength,
            Feature::CorePointSet & cores,
            Feature::DeltaPointSet & deltas)  [protected], [virtual]
```

Read the core points data.

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

Parameters

in,out	*buf*	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last core point data item.
out	*cores*	The set of core data items.
out	*deltas*	The set of delta data items.
in	*dataLength*	The length of the entire ridge count data block.

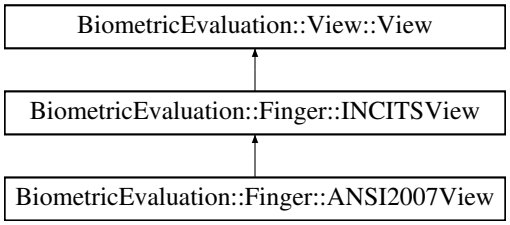
Implements **BiometricEvaluation::Finger::INCITSView** (p. 517).

H.17 BiometricEvaluation::Finger::ANSI2007View Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_ansi2007view.h>`

Inheritance diagram for BiometricEvaluation::Finger::ANSI2007View:



Public Member Functions

- **ANSI2007View** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber)

  Construct an ANSI-2007 finger view from records contained in files.
- **ANSI2007View** (const **Memory::uint8Array** &fmrBuffer, const **Memory::uint8Array** &firBuffer, const uint32_t viewNumber)

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

Public Member Functions inherited from BiometricEvaluation::Finger::INCITSView

- **Feature::INCITSMinutiae getMinutiaeData** () const

  Obtain the set of minutiae records.
- **Finger::Position getPosition** () const

  Obtain the finger position.
- **Finger::Impression getImpressionType** () const

  Obtain the finger impression code.
- uint32_t **getQuality** () const

  Obtain the finger quality value.
- uint16_t **getCaptureEquipmentID** () const

  Obtain the capture equipment identifier.
- bool **isAppendixFCompliant** () const

  Obtain the capture equipment compliance indicator for 'Appendix F'.
- uint16_t **getProductIDOwner** () const

  Obtain the CBEFF product identifier owner.
- uint16_t **getProductIDType** () const

  Obtain the CBEFF product identifier type.
- uint32_t **getRecordLength** () const
- uint8_t **getNumFingerViews** () const
- uint8_t **getFMRReservedByte** () const
- uint32_t **getViewNumber** () const
- uint16_t **getEDBLength** () const
- std::vector< uint8_t > **getMinutiaeReservedData** () const

- void **setMinutiaeData** (const **Feature::INCITSMinutiae** &fmd)

  *Mutator for the **Feature::INCITSMinutiae*** (p. 495) *item.*
- void **setMinutiaeReservedData** (const std::vector< uint8_t > &reservedBits)

  Mutator for the FMD reserved bits vector.

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size** **getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm** **getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Protected Member Functions

- void **readFMRHeader** (**Memory::IndexedBuffer** &buf)
- void **readFVMR** (**Memory::IndexedBuffer** &buf)
- void **readCoreDeltaData** (**Memory::IndexedBuffer** &buf, uint32_t dataLength, Feature::CorePoint↩ Set &cores, Feature::DeltaPointSet &deltas)

  Read the core points data.

Protected Member Functions inherited from BiometricEvaluation::Finger::INCITSView

- **INCITSView** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view↩ Number)

  Construct the common components of an INCITS finger view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &fmrBuffer, const **Memory::uint8Array** &firBuffer, const uint32_t viewNumber)

  Construct an INCITS finger view from records contained in buffers.
- **Memory::uint8Array** const & **getFMRData** () const

  Obtain a reference to the finger minutiae record data buffer.
- **Memory::uint8Array** const & **getFIRData** () const

  Obtain a reference to the finger image record data buffer.
- void **setPosition** (const **Finger::Position** &position)

  Mutator for the position.
- void **setImpressionType** (const **Finger::Impression** &impression)

  Mutator for the impression type.
- void **setQuality** (uint32_t quality)

  Mutator for the finger quality value.

- void **setViewNumber** (uint32_t viewNumber)

  Mutator for the finger view number.
- void **setCaptureEquipmentID** (uint16_t id)

  Mutator for the equipment ID.
- void **setCBEFFProductIDs** (uint16_t owner, uint16_t type)

  Mutator for the CBEFF Product ID owner and type.
- void **setAppendixFCompliance** (bool flag)

  Mutator for the Appendix F compliance indicator.
- void **readFMRHeader** (**Memory::IndexedBuffer** &buf, const uint32_t formatStandard)

  Read the common finger minutiae record header from an INCITS record.
- void **readFVMR** (**Memory::IndexedBuffer** &buf)

  Read the common finger view record information from an INCITS record.
- virtual std::tuple< Feature::MinutiaPointSet, std::vector< uint8_t > > **readMinutiaeDataPoints** (**Memory::IndexedBuffer** &buf, uint32_t count)

  Read the minutiae data points, and extended data blocks.
- virtual void **readExtendedDataBlock** (**Memory::IndexedBuffer** &buf)

  Read the common extended data block.
- virtual Feature::RidgeCountItemSet **readRidgeCountData** (**Memory::IndexedBuffer** &buf, uint32_t dataLength)

  Read the ridge count data.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x30333000

Static Protected Attributes inherited from BiometricEvaluation::Finger::INCITSView

- static const uint32_t **FMR_BASE_FORMAT_ID** = 0x464D5200
- static const uint32_t **ANSI2004_STANDARD** = 1

  The type of record that will be read by the subclass.
- static const uint32_t **ISO2005_STANDARD** = 2
- static const uint32_t **ANSI2007_STANDARD** = 3

Additional Inherited Members

Static Public Member Functions inherited from BiometricEvaluation::Finger::INCITSView

- static **Finger::Position convertPosition** (int incitsFGP)

  Convert a finger postion code from an INCITS finger record to the common code.
- static **Finger::Impression convertImpression** (int incitsIMP)

  Convert a impression type code from an INCITS finger record to the common code.

H.17.1 Detailed Description

A class to represent single finger view and derived information.

A **Finger::ANSI2007View** (p. 274) object represents a finger view from a INCITS/ANSI-2007 **Finger** (p. 122) Minutiae Record.

H.17.2 Constructor & Destructor Documentation

H.17.2.1 ANSI2007View() [1/2]

```
BiometricEvaluation::Finger::ANSI2007View::ANSI2007View (
            const std::string & fmrFilename,
            const std::string & firFilename,
            const uint32_t viewNumber)
```

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

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

Parameters

in	*fmrFilename*	The name of the file containing the complete finger minutiae record.

Parameters

in	*firFilename*	The name of the file containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError *(p. 390)*	Invalid record format.

H.17.2.2 ANSI2007View() [2/2]

```
BiometricEvaluation::Finger::ANSI2007View::ANSI2007View (
            const  Memory::uint8Array & fmrBuffer,
            const  Memory::uint8Array & firBuffer,
            const uint32_t viewNumber)
```

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

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

Parameters

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

Parameters

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

Exceptions

Error::DataError *(p. 390)*	Invalid record format.

H.17.3 Member Function Documentation

H.17.3.1 readCoreDeltaData()

```
void BiometricEvaluation::Finger::ANSI2007View::readCoreDeltaData (
             Memory::IndexedBuffer & buf,
            uint32_t dataLength,
            Feature::CorePointSet & cores,
            Feature::DeltaPointSet & deltas)  [protected], [virtual]
```

Read the core points data.

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

Parameters

in,out	*buf*	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last core point data item.
out	*cores*	The set of core data items.
out	*deltas*	The set of delta data items.
in	*dataLength*	The length of the entire ridge count data block.

Implements **BiometricEvaluation::Finger::INCITSView** (p. 517).

H.18 BiometricEvaluation::Device::Smartcard::APDU Class Reference

Public Attributes

- uint8_t **cla**
- uint8_t **ins**
- uint8_t **p1**
- uint8_t **p2**
- uint16_t **lc**
- uint8_t **nc** [MAX_NC_SIZE]
- uint16_t **le**
- uint8_t **field_mask**

Static Public Attributes

- static const int **FIELD_LC** {0x00000001}
- static const int **FIELD_LE** {0x00000002}
- static const int **FLEN_CLA** {1}
- static const int **FLEN_INS** {1}
- static const int **FLEN_P1** {1}
- static const int **FLEN_P2** {1}
- static const int **FLEN_LC_SHORT** {1}
- static const int **FLEN_LC_EXTENDED** {3}
- static const int **FLEN_LE_SHORT** {1}
- static const int **FLEN_LE_EXTENDED** {3}
- static const int **FLEN_TRAILER** {2}
- static const int **FLAG_CLA_NOCHAIN** {0x00}
- static const int **FLAG_CLA_CHAIN** {0x10}
- static const int **MAX_NC_SIZE** {0xFFFF}
- static const int **MAX_LE_SIZE** {0xFFFF}
- static const int **MAX_SHORT_LC** {255}
- static const int **MAX_SHORT_LE** {255}
- static const int **HEADER_LEN** {FLEN_CLA + FLEN_INS + FLEN_P1 + FLEN_P2}
- static const int **NORMAL_COMPLETE** {0x90}
- static const int **NORMAL_CHAINING** {0x61}
- static const int **WARN_NVM_UNCHANGED** {0x62}
- static const int **WARN_NVM_CHANGED** {0x63}
- static const int **EXEC_ERR_NVM_UNCHANGED** {0x64}
- static const int **EXEC_ERR_NVM_CHANGED** {0x65}
- static const int **EXEC_ERR_SECURITY** {0x66}
- static const int **CHECK_ERR_WRONG_LENGTH** {0x67}
- static const int **CHECK_ERR_CLA_FUNCTION** {0x68}
- static const int **CHECK_ERR_CMD_NOT_ALLOWED** {0x69}
- static const int **CHECK_ERR_WRONG_PARAM_QUAL** {0x6A}
- static const int **CHECK_ERR_WRONG_PARAM** {0x6B}
- static const int **CHECK_ERR_WRONG_LE** {0x6C}
- static const int **CHECK_ERR_INVALID_INS** {0x6D}
- static const int **CHECK_ERR_CLA_UNSUPPORTED** {0x6E}

- static const int **CHECK_ERR_NO_DIAGNOSIS** {0x6F}
- static const int **NO_INFORMATION** {0x00}
- static const int **INCORRECT_PARAMETERS** {0x80}
- static const int **FUNCTION_NOT_SUPPORTED** {0x81}
- static const int **FILE_OR_APP_NOT_FOUND** {0x82}
- static const int **RETRY_COUNTER_MASK** {0x0F}
- static const int **RETRY_COUNTER_INDICATOR** {0xC0}
- static const int **RETRY_COUNTER_INDICATOR_MASK** {0xF0}
- static const int **RETRY_COUNTER_MAX** {15}

H.18.1 Member Data Documentation

H.18.1.1 cla

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::cla`

The class byte

H.18.1.2 FIELD_LC

`const int BiometricEvaluation::Device::Smartcard::APDU::FIELD_LC {0x00000001} [static]`

Lc field is present; Implies Nc present as well

H.18.1.3 FIELD_LE

`const int BiometricEvaluation::Device::Smartcard::APDU::FIELD_LE {0x00000002} [static]`

Le field is present, response data expected

H.18.1.4 field_mask

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::field_mask`

Mask of optional fields; use field bit masks

H.18.1.5 ins

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::ins`

Instruction byte

H.18.1.6 lc

`uint16_t BiometricEvaluation::Device::Smartcard::APDU::lc`

Lc, length of the Nc field

H.18.1.7 le

`uint16_t BiometricEvaluation::Device::Smartcard::APDU::le`

Le, expected response length

H.18.1.8 nc

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::nc[MAX_NC_SIZE]`

Nc, command data

H.18.1.9 p1

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::p1`

P1 byte

H.18.1.10 p2

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::p2`

P2 byte

H.19 BiometricEvaluation::Device::Smartcard::APDUException Struct Reference

Exception thrown when a command fails.

`#include <be_device_smartcard.h>`

Public Member Functions

- **APDUException** ()=default
- **APDUException** (const **APDUResponse** & **response**, const **Memory::uint8Array** & **apdu**)

Public Attributes

- **APDUResponse response**
- **Memory::uint8Array apdu**

H.19.1 Detailed Description

Exception thrown when a command fails.

This object is thrown when the status words returned from the card indicate an error occurred when a command was sent to the card. Any data returned by the card and the **APDU** (p. 281) that was sent are contained within this object.

H.19.2 Constructor & Destructor Documentation

H.19.2.1 APDUException() [1/2]

```
BiometricEvaluation::Device::Smartcard::APDUException::APDUException () [default]
```

Constructor.

H.19.2.2 APDUException() [2/2]

```
BiometricEvaluation::Device::Smartcard::APDUException::APDUException (
            const APDUResponse & response,
            const Memory::uint8Array & apdu)
```

Constructor.

Parameters

repines	The partial response data and status
apdu	The raw **APDU** (p. 281) that was sent.

H.19.3 Member Data Documentation

H.19.3.1 apdu

Memory::uint8Array BiometricEvaluation::Device::Smartcard::APDUException::apdu

The raw **APDU** (p. 281) that was sent.

H.19.3.2 response

APDUResponse BiometricEvaluation::Device::Smartcard::APDUException::response

The partial response data and status words from the failed command.

H.20 BiometricEvaluation::Device::Smartcard::APDUResponse Struct Reference

The data and status words returned by the card in response to a command.

`#include <be_device_smartcard.h>`

Public Member Functions

- **APDUResponse** ()=default
- **APDUResponse** (const **Memory::uint8Array** & **data**, const uint8_t **sw1**, const uint8_t **sw2**)

Public Attributes

- uint8_t **sw1** {0}
- uint8_t **sw2** {0}
- **Memory::uint8Array data**

H.20.1 Detailed Description

The data and status words returned by the card in response to a command.

H.20.2 Constructor & Destructor Documentation

H.20.2.1 APDUResponse() [1/2]

```
BiometricEvaluation::Device::Smartcard::APDUResponse::APDUResponse ()  [default]
```

Constructor

H.20.2.2 APDUResponse() [2/2]

```
BiometricEvaluation::Device::Smartcard::APDUResponse::APDUResponse (
            const  Memory::uint8Array & data,
            const uint8_t sw1,
            const uint8_t sw2)
```

Constructor

Parameters

data	The response data; may be empty.
sw1	Status word one.
sw2	Status word two.

H.20.3 Member Data Documentation

H.20.3.1 data

```
Memory::uint8Array BiometricEvaluation::Device::Smartcard::APDUResponse::data
```

The response data, possibly incomplete

H.20.3.2 sw1

```
uint8_t BiometricEvaluation::Device::Smartcard::APDUResponse::sw1 {0}
```

status word one

H.20.3.3 sw2

```
uint8_t BiometricEvaluation::Device::Smartcard::APDUResponse::sw2 {0}
```

status word two

H.21 BiometricEvaluation::Framework::API< T > Class Template Reference

A convenient way to execute biometric technology evaluation **API** (p. 286) methods safely.

`#include <be_framework_api.h>`

Classes

- class **Result**

Public Member Functions

- **API** ()
- **Result** **call** (const std::function< T(void)> &operation, const std::function< void(const **Result** &)> &success={}, const std::function< void(const **Result** &)> &failure={})

  *Invoke an operation. @detail Invoking operations within this method implicitly wraps the operation in a SignalManager, Watchdog, and Timer, and follows evaluation best practices for calling an **API** (p. 286) operation.*
- bool **protectionsEnabled** () const

  *Obtain whether or not **all** protections enabled by this object are enabled.*
- void **setProtectionsEnabled** (const bool **protectionsEnabled**)

  Wholesale change of process protections enabled by this object.
- bool **willRethrowExceptions** () const

  *Obtain whether or not exceptions caught in **call()** (p. 287) will be rethrown.*
- void **setRethrowExceptions** (const bool shouldRethrow)

  *Change whether or not exceptions caught in **call()** (p. 287) should be rethrown.*
- void **setCatchExceptions** (const bool catchExceptions)

  *Set whether or not to catch exceptions from **call()** (p. 287), triggering the `failure` block.*
- bool **willCatchExceptions** () const

  *Obtain whether or not exceptions raised in **call()** (p. 287) will be caught, triggering the `failure` block.*
- std::shared_ptr< **BiometricEvaluation::Time::Timer** > **getTimer** () noexcept

  Obtain the timer object.
- std::shared_ptr< **BiometricEvaluation::Time::Watchdog** > **getWatchdog** () noexcept

  Obtain the watchdog timer object.
- std::shared_ptr< **BiometricEvaluation::Error::SignalManager** > **getSignalManager** () noexcept

  Obtain the signal manager object.

H.21.1 Detailed Description

template<typename T>
class BiometricEvaluation::Framework::API< T >

A convenient way to execute biometric technology evaluation **API** (p. 286) methods safely.

Note

One **API** (p. 286) object should be instantiated per process/thread.

H.21.2 Constructor & Destructor Documentation

H.21.2.1 API()

```
template<typename T >
BiometricEvaluation::Framework::API< T >::API ()
```

Constructor

H.21.3 Member Function Documentation

H.21.3.1 call()

```
template<typename T >
BiometricEvaluation::Framework::API< T >::Result  BiometricEvaluation::Framework::API< T >↩
::call (
            const std::function< T(void)> & operation,
            const std::function< void(const  Result &)> & success = {},
            const std::function< void(const  Result &)> & failure = {})
```

Invoke an operation. @detail Invoking operations within this method implicitly wraps the operation in a SignalManager, Watchdog, and Timer, and follows evaluation best practices for calling an **API** (p. 286) operation.

Parameters

operation	A reference to a function that returns a **Status** (p. 786). (i.e., an **API** (p. 286) method).
success	Operations invoked if operation returns.
failure	Operations invoked if we abort the operation.

Parameters

rethrowExceptions	Whether or not to rethrow an exception caught from `operation`.

Returns

Analytics about the return of operation.

Exceptions

...	Exceptions raised from `operation`, if caught (**willCatchExceptions()** (p. 291) is `true`), are rethrown when **API::willRetl**

Note

success is called and currentState == **APICurrentState::Completed** (p. **??**) if operation returns, regardless of the Code of operation's **Status** (p. 786).

H.21.3.2 getSignalManager()

```
template<typename T >
std::shared_ptr< BiometricEvaluation::Error::SignalManager > BiometricEvaluation::Framework↩
::API< T >::getSignalManager () [inline], [noexcept]
```

Obtain the signal manager object.

Returns

Signal manager object.

H.21.3.3 getTimer()

```
template<typename T >
std::shared_ptr< BiometricEvaluation::Time::Timer > BiometricEvaluation::Framework::API< T
>::getTimer () [inline], [noexcept]
```

Obtain the timer object.

Returns

Timer object.

H.21.3.4 getWatchdog()

```
template<typename T >
std::shared_ptr< BiometricEvaluation::Time::Watchdog > BiometricEvaluation::Framework::API<
T >::getWatchdog () [inline], [noexcept]
```

Obtain the watchdog timer object.

Returns

Watchdog timer object.

H.21.3.5 protectionsEnabled()

```
template<typename T >
bool BiometricEvaluation::Framework::API< T >::protectionsEnabled () const [inline]
```

Obtain whether or not **all** protections enabled by this object are enabled.
Protections include:

- Catching exceptions
- Not rethrowing exceptions
- Enabling WatchdogTimer
- Enabling SignalManager

Returns

`true` if **all** protections are enabled, `false` if one or more protections are disabled.

Note

Individual protection statuses may be queried through their respective objects/methods.

H.21.3.6 setCatchExceptions()

```
template<typename T >
void BiometricEvaluation::Framework::API< T >::setCatchExceptions (
            const bool catchExceptions) [inline]
```

Set whether or not to catch exceptions from **call()** (p. 287), triggering the `failure` block.

Parameters

catchExceptions	`true` if **call**() (p. 287)'s `operation` should be executed within a try block, `false` otherwise.

H.21.3.7 setProtectionsEnabled()

```
template<typename T >
void  BiometricEvaluation::Framework::API< T >::setProtectionsEnabled (
            const bool protectionsEnabled)  [inline]
```

Wholesale change of process protections enabled by this object.
Protections include:

- Catching exceptions
- Not rethrowing exceptions
- Enabling WatchdogTimer
- Enabling SignalManager

Parameters

protectionsEnabled	`true` if **all** protections should be enabled, `false` if **all** protections should be disabled.

Note

Protections can be enabled or disabled individually through their respective objects/methods.

H.21.3.8 setRethrowExceptions()

```
template<typename T >
void  BiometricEvaluation::Framework::API< T >::setRethrowExceptions (
            const bool shouldRethrow)  [inline]
```

Change whether or not exceptions caught in **call()** (p. 287) should be rethrown.

Parameters

shouldRethrow	`true` if exceptions raised in **call()** (p. 287) will be rethrown, `false` otherwise.

Note

Exceptions will not be caught (and thus not rethrown) if **willCatchExceptions()** (p. 291) is `false`.

H.21.3.9 willCatchExceptions()

```
template<typename T >
bool  BiometricEvaluation::Framework::API< T >::willCatchExceptions () const  [inline]
```

Obtain whether or not exceptions raised in **call()** (p. 287) will be caught, triggering the `failure` block.

Returns

`true` if **call()** (p. 287)'s `operation` will be executed within a try block, `false` otherwise.

H.21.3.10 willRethrowExceptions()

```
template<typename T >
bool  BiometricEvaluation::Framework::API< T >::willRethrowExceptions () const  [inline]
```

Obtain whether or not exceptions caught in **call()** (p. 287) will be rethrown.

Returns

`true` if exceptions raised in **call()** (p. 287) will be rethrown, `false` otherwise.

Note

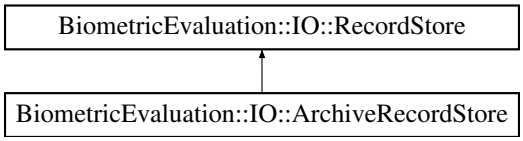
Exceptions will not be caught (and thus not rethrown) if **willCatchExceptions()** (p. 291) is `false`.

H.22 BiometricEvaluation::IO::ArchiveRecordStore Class Reference

This class implements the **IO::RecordStore** (p. 700) interface by storing data items in single file, with an associated manifest file.

`#include <be_io_archiverecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ArchiveRecordStore:



Public Member Functions

- **ArchiveRecordStore** (const std::string &pathname, const std::string &description)
- **ArchiveRecordStore** (const std::string &pathname, **IO::Mode** mode= **IO::Mode::ReadOnly**)
- ~**ArchiveRecordStore** ()
- void **sync** () const override
- void **insert** (const std::string &key, const void *const data, const uint64_t size) override
- void **remove** (const std::string &key) override
- **Memory::uint8Array** **read** (const std::string &key) const override

  Read a complete record from a store.
- uint64_t **length** (const std::string &key) const override
- void **flush** (const std::string &key) const override
- **RecordStore::Record** **sequence** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  *Move the **RecordStore** (p. 700).*
- uint64_t **getSpaceUsed** () const override

  Obtain real storage utilization.
- unsigned int **getCount** () const override
- std::string **getPathname** () const override
- std::string **getDescription** () const override
- void **changeDescription** (const std::string &description) override
- bool **needsVacuum** ()
- std::string **getArchiveName** () const
- std::string **getManifestName** () const
- **ArchiveRecordStore** (const **ArchiveRecordStore** &)=delete
- **ArchiveRecordStore** & **operator=** (const **ArchiveRecordStore** &)=delete
- virtual void **insert** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const void *const data, const uint64_t size)

Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- virtual bool **containsKey** (const std::string &key) const

  *Determines whether the **RecordStore** (p. 700) contains an element with the specified key.*
- virtual **iterator** **begin** () noexcept
- virtual **iterator** **end** () noexcept

Static Public Member Functions

- static bool **needsVacuum** (const std::string &pathname)
- static void **vacuum** (const std::string &pathname)

Static Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- static bool **isRecordStore** (const std::string &pathname)

  *Determine if a location appears to be a **RecordStore** (p. 700).*
- static std::shared_ptr< **RecordStore** > **openRecordStore** (const std::string &pathname, **IO::Mode** mode= **Mode::ReadOnly**)

  *Open an existing **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static std::shared_ptr< **RecordStore** > **createRecordStore** (const std::string &pathname, const std::string &description, const **IO::RecordStore::Kind** &kind)

  *Create a new **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static void **removeRecordStore** (const std::string &pathname)
- static void **mergeRecordStores** (const std::string &mergePathname, const std::string &description, const **IO::RecordStore::Kind** &kind, const std::vector< std::string > &pathnames, const std::function< bool()> &interrupt=[ ]() {return(false);})

  *Create a new **RecordStore** (p. 700) that contains the contents of several other RecordStores.*

Static Public Attributes

- static const std::string **MANIFEST_FILE_NAME**
- static const std::string **ARCHIVE_FILE_NAME**
- static const long **OFFSET_RECORD_REMOVED** = -1

Static Public Attributes inherited from BiometricEvaluation::IO::RecordStore

- static const std::string **INVALIDKEYCHARS**
- static const int **BE_RECSTORE_SEQ_START** = 1
- static const int **BE_RECSTORE_SEQ_NEXT** = 2

Additional Inherited Members

Public Types inherited from BiometricEvaluation::IO::RecordStore

- enum class **Kind** {
  BerkeleyDB , **Archive** , **File** , **SQLite** ,
  Compressed , **List** , **Default** = BerkeleyDB }
- using **Record** = struct Record
- using **iterator** = **IO::RecordStoreIterator**

H.22.1 Detailed Description

This class implements the **IO::RecordStore** (p. 700) interface by storing data items in single file, with an associated manifest file.

Archives consist of binary records written back to back of each other. To pull information out of an archive, a manifest file is written in the same directory as the archive file.

Each record is assigned a string key, which will be required for retrieving the data. As the data is written, a plain text entry is entered into the manifest in the format:
key offset size
where offset is the offset into the archive file key's data chunk resides and size is the length of key's data chunk.

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

H.22.2 Constructor & Destructor Documentation

H.22.2.1 ArchiveRecordStore() [1/2]

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

Create a new **ArchiveRecordStore** (p. 292), read/write mode.

Parameters

in	*pathname*	The directory where the store is to be created.
in	*description*	The store's description.

Exceptions

Error::ObjectExists (p. 637)	The store already exists.
Error::StrategyError (p. 789)	An error occurred when accessing the underlying file system.

H.22.2.2 ArchiveRecordStore() [2/2]

```
BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore (
            const std::string & pathname,
             IO::Mode mode =  IO::Mode::ReadOnly)
```

Open an existing **ArchiveRecordStore** (p. 292).

Parameters

in	*pathname*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The store does not exist.
Error::StrategyError (p. 789)	An error occurred when accessing the underlying file system.

H.22.2.3 ~ArchiveRecordStore()

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

Destructor.

H.22.3 Member Function Documentation

H.22.3.1 changeDescription()

```
void BiometricEvaluation::IO::ArchiveRecordStore::changeDescription (
            const std::string & description)  [override], [virtual]
```

Change the description of the **RecordStore** (p. 700).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 703).

H.22.3.2 flush()

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

Commit the record's data to storage.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 704).

H.22.3.3 getArchiveName()

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

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

Returns

Path to archive file.

H.22.3.4 getCount()

```
unsigned int BiometricEvaluation::IO::ArchiveRecordStore::getCount () const  [override], [virtual]
```

Obtain the number of items in the **RecordStore** (p. 700).

Returns

The number of items in the **RecordStore** (p. 700).

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.22.3.5 getDescription()

```
std::string BiometricEvaluation::IO::ArchiveRecordStore::getDescription () const  [override],
[virtual]
```

Obtain a textual description of the **RecordStore** (p. 700).

Returns

The **RecordStore** (p. 700)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.22.3.6 getManifestName()

```
std::string BiometricEvaluation::IO::ArchiveRecordStore::getManifestName () const
```

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

Returns

Path to manifest file.

H.22.3.7 getPathname()

```
std::string BiometricEvaluation::IO::ArchiveRecordStore::getPathname () const [override], [virtual]
```

Return the path name of the **RecordStore** (p. 700).

Returns

Where in the file system the **RecordStore** (p. 700) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.22.3.8 getSpaceUsed()

```
uint64_t BiometricEvaluation::IO::ArchiveRecordStore::getSpaceUsed () const [override], [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** (p. 700).

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 706).

H.22.3.9 insert() [1/2]

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

Insert a record into the store.

Parameters

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

in	*data*	The data for the record.

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 706).

H.22.3.10 insert() [2/2]

```
void BiometricEvaluation::IO::ArchiveRecordStore::insert (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [override], [virtual]
```

Insert a record into the store.

Parameters

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

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Implements **BiometricEvaluation::IO::RecordStore** (p. 707).

H.22.3.11 length()

```
uint64_t BiometricEvaluation::IO::ArchiveRecordStore::length (
            const std::string & key) const  [override], [virtual]
```

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 708).

H.22.3.12 move()

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

Move the **RecordStore** (p. 700).

The **RecordStore** (p. 700) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 700).

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 710).

H.22.3.13 needsVacuum() [1/2]

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

See if the **ArchiveRecordStore** (p. 292) would benefit from calling **vacuum()** (p. 305) to remove deleted entries, since **vacuum()** (p. 305) is an expensive operation.

Returns

true if **vacuum()** (p. 305) would be beneficial false otherwise

H.22.3.14 needsVacuum() [2/2]

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

See if the **ArchiveRecordStore** (p. 292) would benefit from calling **vacuum()** (p. 305) to remove deleted entries, since **vacuum()** (p. 305) is an expensive operation.

Parameters

in	*pathname*	The path name of the existing **RecordStore** (p. 700).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record with the given key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Returns

true if **vacuum()** (p. 305) would be beneficial false otherwise

H.22.3.15 read()

```
Memory::uint8Array BiometricEvaluation::IO::ArchiveRecordStore::read (
            const std::string & key) const  [override], [virtual]
```

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

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

Returns

The record associated with the key.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 712).

H.22.3.16 remove()

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

Remove a record from the store.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 713).

H.22.3.17 replace() [1/2]

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

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	A record for the key does not exist.
Error::StrategyError *(p. 789)*	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.22.3.18 replace() [2/2]

```
virtual void BiometricEvaluation::IO::RecordStore::replace (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [virtual]
```

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.22.3.19 sequence()

```
RecordStore::Record BiometricEvaluation::IO::ArchiveRecordStore::sequence (
            int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the function to return the next record. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The record that is currently in sequence.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 715).

H.22.3.20 sequenceKey()

```
std::string BiometricEvaluation::IO::ArchiveRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key.

Sequencing means to start at some point in the store and return the key, then repeatedly calling the function to return the next key. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The key of the currently sequenced record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 716).

H.22.3.21 setCursorAtKey()

```
void BiometricEvaluation::IO::ArchiveRecordStore::setCursorAtKey (
            const std::string & key)  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 700), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 303).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 303).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.22.3.22 sync()

```
void BiometricEvaluation::IO::ArchiveRecordStore::sync () const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.22.3.23 vacuum()

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

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

Parameters

in	*pathname*	The pathname of the existing **RecordStore** (p. 700).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record with the given key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Note

This is an expensive operation.

H.22.4 Member Data Documentation

H.22.4.1 ARCHIVE_FILE_NAME

```
const std::string BiometricEvaluation::IO::ArchiveRecordStore::ARCHIVE_FILE_NAME  [static]
```

Name of the archive file on disk

H.22.4.2 MANIFEST_FILE_NAME

`const std::string BiometricEvaluation::IO::ArchiveRecordStore::MANIFEST_FILE_NAME [static]`

Name of the manifest file on disk

H.22.4.3 OFFSET_RECORD_REMOVED

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

Offset placeholder indicating a removed record

H.23 BiometricEvaluation::Memory::AutoArray< T > Class Template Reference

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

`#include <be_memory_autoarray.h>`

Public Types

- using **value_type** = T
- using **size_type** = size_t
- using **iterator** = **AutoArrayIterator**<false, T>
- using **const_iterator**
- using **reference** = T&
- using **const_reference** = const T&

Public Member Functions

- **operator T*** ()

  *Convert **AutoArray** (p. 306) to T array.*
- **operator const T** * () const

  *Convert **AutoArray** (p. 306) to const T array.*
- **reference operator[ ]** (ptrdiff_t index)

  Subscripting operator overload with unchecked access.
- **const_reference operator[ ]** (ptrdiff_t index) const

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

  *Subscript into the **AutoArray** (p. 306) with checked access.*
- **const_reference at** (ptrdiff_t index) const

  *Subscript into the **AutoArray** (p. 306) with checked access.*
- **iterator begin** ()

  *Obtain an iterator to the beginning of the **AutoArray** (p. 306).*
- **const_iterator begin** () const

  *Obtain an iterator to the beginning of the **AutoArray** (p. 306).*
- **const_iterator cbegin** () const

  *Obtain an iterator to the beginning of the **AutoArray** (p. 306).*
- **iterator end** ()

  *Obtain an iterator to the end of the **AutoArray** (p. 306).*
- **const_iterator end** () const

*Obtain an iterator to the end of the **AutoArray*** (p. 306).

- **const_iterator cend** () const

  *Obtain an iterator to the end of the **AutoArray*** (p. 306).

- **size_type size** () const

  Obtain the number of accessible elements.

- void **resize** (**size_type** new_size, bool free=false)

  Change the number of accessible elements.

- void **copy** (const T *buffer)

  *Deep-copy the contents of a buffer into this **AutoArray*** (p. 306).

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

  *Deep-copy the contents of a buffer into this **AutoArray*** (p. 306).

- std::vector< T > **to_vector** () const

  *Obtain a copy of elements in this **AutoArray*** (p. 306) *as a vector.*

- **AutoArray** (**size_type size**=0)

  *Construct an **AutoArray*** (p. 306).

- **AutoArray** (const **AutoArray** & **copy**)

  *Construct an **AutoArray*** (p. 306).

- **AutoArray** (**AutoArray** &&rvalue) noexcept

  *Construct an **AutoArray*** (p. 306).

- **AutoArray** (std::initializer_list< T > ilist)

  *Construct an **AutoArray*** (p. 306).

- **AutoArray** & **operator=** (const **AutoArray** &other)

  Copy assignment operator overload performing a deep copy.

- **AutoArray** & **operator=** (**AutoArray** &&other) noexcept(noexcept(std::swap(std::declval< **value_type** & >(), std::declval< **value_type** & >())) &&noexcept(std::swap(std::declval< **size_type** & >(), std::declval< **size_type** & >())))

  Move assignment operator.

- ~**AutoArray** ()

H.23.1 Detailed Description

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

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

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

Forward declaration.

H.23.2 Member Typedef Documentation

H.23.2.1 const_iterator

```
template<typename T >
using BiometricEvaluation::Memory::AutoArray< T >::const_iterator
```

Initial value:

```
AutoArrayIterator<true, T>
```

Const iterator of element

H.23.2.2 const_reference

```
template<typename T >
using BiometricEvaluation::Memory::AutoArray< T >::const_reference = const T&
```

Const reference element

H.23.2.3 iterator

```
template<typename T >
using BiometricEvaluation::Memory::AutoArray< T >::iterator = AutoArrayIterator<false, T>
```

Iterator of element

H.23.2.4 reference

```
template<typename T >
using BiometricEvaluation::Memory::AutoArray< T >::reference = T&
```

Reference to element

H.23.2.5 size_type

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

Type of subscripts, counts, etc.

H.23.2.6 value_type

```
template<typename T >
using BiometricEvaluation::Memory::AutoArray< T >::value_type = T
```

Type of element

H.23.3 Constructor & Destructor Documentation

H.23.3.1 AutoArray() [1/4]

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

Construct an **AutoArray** (p. 306).

Parameters

in	*size*	The number of elements this **AutoArray** (p. 306) should initially hold.

Exceptions

Error::MemoryError (p. 604)	Could not allocate new memory.

H.23.3.2 AutoArray() [2/4]

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

Construct an **AutoArray** (p. 306).

Parameters

in	*copy*	An **AutoArray** (p. 306) whose contents will be deep copied into the new **AutoArray** (p. 306).

Exceptions

Error::MemoryError (p. 604)	Could not allocate new memory.

H.23.3.3 AutoArray() [3/4]

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

Construct an **AutoArray** (p. 306).

Parameters

in	*rvalue*	An rvalue reference to an **AutoArray** (p. 306) whose contents will be moved and destroyed.

H.23.3.4 AutoArray() [4/4]

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

Construct an **AutoArray** (p. 306).

Parameters

in	*ilist*	An initializer list of type T.

H.23.3.5 ~AutoArray()

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

Destructor

H.23.4 Member Function Documentation

H.23.4.1 at() [1/2]

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

Subscript into the **AutoArray** (p. 306) 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** (p. 306).

H.23.4.2 at() [2/2]

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T > ::const_reference BiometricEvaluation::Memory↩
::AutoArray< T >::at (
            ptrdiff_t index) const
```

Subscript into the **AutoArray** (p. 306) 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** (p. 306).

H.23.4.3 begin() [1/2]

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

Obtain an iterator to the beginning of the **AutoArray** (p. 306).

Returns

Iterator positioned at the first element of the **AutoArray** (p. 306).

H.23.4.4 begin() [2/2]

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

Obtain an iterator to the beginning of the **AutoArray** (p. 306).

Returns

Const iterator positioned at the first element of the **AutoArray** (p. 306).

H.23.4.5 cbegin()

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T > ::const_iterator BiometricEvaluation::Memory::↩
AutoArray< T >::cbegin () const
```

Obtain an iterator to the beginning of the **AutoArray** (p. 306).

Returns

Const iterator positioned at the first element of the **AutoArray** (p. 306).

H.23.4.6 cend()

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T > ::const_iterator BiometricEvaluation::Memory::↩
AutoArray< T >::cend () const
```

Obtain an iterator to the end of the **AutoArray** (p. 306).

Returns

Iterator positioned at the one-past-last element of the **AutoArray** (p. 306).

H.23.4.7 copy() [1/2]

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

Deep-copy the contents of a buffer into this **AutoArray** (p. 306).

Parameters

in	*buffer*	An allocated buffer whose contents will be deep-copied into this object. Only **size()** (p. 318) bytes will be copied.

Warning

If buffer is smaller in size than the current size of the **AutoArray** (p. 306), you MUST call **copy(const T*, size_type)** (p. 313). This method must only be used when buffer is larger than or equal to the size of the **AutoArray** (p. 306).

H.23.4.8 copy() [2/2]

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

Deep-copy the contents of a buffer into this **AutoArray** (p. 306).

Parameters

in	*buffer*	An allocated buffer whose contents will be deep-copied into this object.

Parameters

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.

H.23.4.9 end() [1/2]

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

Obtain an iterator to the end of the **AutoArray** (p. 306).

Returns

Iterator positioned at the one-past-last element of the **AutoArray** (p. 306).

H.23.4.10 end() [2/2]

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

Obtain an iterator to the end of the **AutoArray** (p. 306).

Returns

Iterator positioned at the one-past-last element of the **AutoArray** (p. 306).

H.23.4.11 operator const T *()

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

Convert **AutoArray** (p. 306) to const T array.

Returns

Const pointer to the beginning of the underlying array storage.

H.23.4.12 operator T*()

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

Convert **AutoArray** (p. 306) to T array.

Returns

Pointer to the beginning of the underlying array storage.

H.23.4.13 operator=() [1/2]

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

Move assignment operator.

Parameters

in	*other*	rvalue reference to another **AutoArray** (p. 306), whose contents will be moved and cleared from itself.

Returns

Reference to the lvalue **AutoArray** (p. 306).

H.23.4.14 operator=() [2/2]

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

Copy assignment operator overload performing a deep copy.

Parameters

in	*other*	**AutoArray** (p. 306) to be copied.

Returns

Reference to a new **AutoArray** (p. 306) object, the lvalue **AutoArray** (p. 306).

Exceptions

Error::MemoryError *(p. 604)*	Could not allocate new memory.

H.23.4.15 operator[]() [1/2]

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

Subscripting operator overload with unchecked access.

Parameters

in	*index*	Subscript into underlying storage.

Returns

Reference to the element at the specified index.

H.23.4.16 operator[]() [2/2]

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

Const subscripting operator overload with unchecked access.

Parameters

in	*index*	Subscript into underlying storage.

Returns

Const reference to the element at the specified index.

H.23.4.17 resize()

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

Change the number of accessible elements.

Parameters

in	*new_size*	The number of elements the **AutoArray** (p. 306) 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 *(p. 604)*	Problem allocating memory.

H.23.4.18 size()

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T > ::size_type BiometricEvaluation::Memory::Auto↩
Array< T >::size () const
```

Obtain the number of accessible elements.

Returns

Number of accessible elements.

Note

If **resize()** (p. 317) has been called, the value returned from **size()** (p. 318) may be smaller than the actual allocated size of the underlying storage.

H.23.4.19 to_vector()

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

Obtain a copy of elements in this **AutoArray** (p. 306) as a vector.

Warning

A key difference between vectors and AutoArrays is that all elements of a vector must be initialized. Calling this method on an **AutoArray** (p. 306) where not all elements have been initialized will likely cause undefined behavior.

Returns

A vector containing the contents of this **AutoArray** (p. 306).

H.24 BiometricEvaluation::Memory::AutoArrayIterator< C, T > Class Template Reference

RandomAccessIterator for any **AutoArray** (p. 306).

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

Public Types

- using **iterator_category**
- using **value_type**
- using **difference_type** = std::ptrdiff_t
- using **pointer**
- using **reference**
- using **container**

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

Public Member Functions

- **AutoArrayIterator** (**container** autoArray=nullptr, **difference_type** offset=0)

  Default constructor.
- **AutoArrayIterator** (const **AutoArrayIterator** &rhs)=default
- **AutoArrayIterator** (**AutoArrayIterator** &&rhs)=default
- ~**AutoArrayIterator** ()=default
- **AutoArrayIterator** & **operator=** (**pointer** rhs)
- **AutoArrayIterator** & **operator=** (const **AutoArrayIterator** &rhs)=default
- **AutoArrayIterator** & **operator+=** (const **difference_type** &rhs)
- **AutoArrayIterator** & **operator-=** (const **difference_type** &rhs)
- **reference** **operator*** () const
- **pointer** **operator->** () const
- **reference** **operator[ ]** (const **difference_type** &rhs) const
- **AutoArrayIterator** & **operator++** ()
- **AutoArrayIterator** & **operator--** ()
- **AutoArrayIterator** **operator++** (int)
- **AutoArrayIterator** **operator--** (int)
- **AutoArrayIterator** **operator+** (const **AutoArrayIterator** &rhs) const
- **difference_type** **operator-** (const **AutoArrayIterator**< C, T > &rhs) const
- **AutoArrayIterator** **operator+** (const **difference_type** &rhs) const
- **AutoArrayIterator** **operator-** (const **difference_type** &rhs) const
- bool **operator==** (const **AutoArrayIterator** &rhs) const
- bool **operator!=** (const **AutoArrayIterator** &rhs) const
- bool **operator>** (const **AutoArrayIterator** &rhs) const
- bool **operator<** (const **AutoArrayIterator** &rhs) const
- bool **operator>=** (const **AutoArrayIterator** &rhs) const
- bool **operator<=** (const **AutoArrayIterator** &rhs) const

Friends

- **AutoArrayIterator** **operator+** (const **difference_type** &lhs, const **AutoArrayIterator** &rhs)
- **AutoArrayIterator** **operator-** (const **difference_type** &lhs, const **AutoArrayIterator** &rhs)

H.24.1 Detailed Description

template<bool C, class T>
class BiometricEvaluation::Memory::AutoArrayIterator< C, T >

RandomAccessIterator for any **AutoArray** (p. 306).

Note

This class encapsulates a const and non-const iterator in one. The first parameter to the template is a boolean whether or not to use the const version of the iterator. The second is the contained type of the **AutoArray** (p. 306).

H.24.2 Member Typedef Documentation

H.24.2.1 container

```
template<bool C, class T >
using BiometricEvaluation::Memory::AutoArrayIterator< C, T >::container
```

Initial value:

```
typename std::conditional<C,
            const AutoArray<T>*, AutoArray<T>*>::type
```

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

H.24.2.2 difference_type

```
template<bool C, class T >
using BiometricEvaluation::Memory::AutoArrayIterator< C, T >::difference_type = std::ptrdiff_t
```

Type used to measure distance between iterators

H.24.2.3 iterator_category

```
template<bool C, class T >
using BiometricEvaluation::Memory::AutoArrayIterator< C, T >::iterator_category
```

Initial value:

```
            std::random_access_iterator_tag
```

Type of iterator

H.24.2.4 pointer

```
template<bool C, class T >
using BiometricEvaluation::Memory::AutoArrayIterator< C, T >::pointer
```

Initial value:

```
typename
            std::conditional<C, const T*, T*>::type
```

Pointer to the type iterated over

H.24.2.5 reference

```
template<bool C, class T >
using BiometricEvaluation::Memory::AutoArrayIterator< C, T >::reference
```

Initial value:

```
typename
            std::conditional<C, const T&, T&>::type
```

Reference to the type iterated over

H.24.2.6 value_type

```
template<bool C, class T >
using BiometricEvaluation::Memory::AutoArrayIterator< C, T >::value_type
```

Initial value:

```
typename
            std::conditional<C, const T, T>::type
```

Type when dereferencing iterators

H.24.3 Constructor & Destructor Documentation

H.24.3.1 AutoArrayIterator() [1/3]

```
template<bool C, class T >
BiometricEvaluation::Memory::AutoArrayIterator< C, T >::AutoArrayIterator (
            container autoArray = nullptr,
            difference_type offset = 0)  [inline]
```

Default constructor.

Parameters

autoArray	Pointer to the **AutoArray** (p. 306) to iterate
offset	The offset into the **AutoArray** (p. 306) where this iterator should start.

H.24.3.2 AutoArrayIterator() [2/3]

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

Default copy constructor

H.24.3.3 AutoArrayIterator() [3/3]

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

Default move constructor

H.24.3.4 ~AutoArrayIterator()

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

Default destructor

H.24.4 Member Function Documentation

H.24.4.1 operator"!=()

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

Returns

Whether or not the offsets are different.

H.24.4.2 operator*()

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

Returns

Object at the current offset.

H.24.4.3 operator+() [1/2]

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

Returns

This object with offset incremented by rhs' offset.

H.24.4.4 operator+() [2/2]

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

Returns

This object with offset incremented rhs.

H.24.4.5 operator++() [1/2]

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

Returns

This object with incremented offset.

H.24.4.6 operator++() [2/2]

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

Returns

This object before incrementing offset.

H.24.4.7 operator+=()

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

Returns

This object with rhs added to offset.

H.24.4.8 operator-() [1/2]

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

Returns

Offset decremented by rhs' offset.

H.24.4.9 operator-() [2/2]

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

Returns

This object with offset decremented rhs.

H.24.4.10 operator--() [1/2]

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

Returns

This object with decremented offset.

H.24.4.11 operator--() [2/2]

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

Returns

This object before decrementing offset.

H.24.4.12 operator-=()

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

Returns

This object with rhs removed from offset.

H.24.4.13 operator->()

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

Returns

Address of object at the current offset.

H.24.4.14 operator<()

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

Returns

true if this offset is < rhs'.

H.24.4.15 operator<=()

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

Returns

true if this offset is <= rhs'.

H.24.4.16 operator=() [1/2]

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

Default assignment operator.

H.24.4.17 operator=() [2/2]

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

Returns

This object with offset set to rhs.

H.24.4.18 operator==()

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

Returns

Whether or not the offsets are the same.

H.24.4.19 operator>()

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

Returns

true if this offset is > rhs'.

H.24.4.20 operator>=()

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

Returns

true if this offset is >= rhs'.

H.24.4.21 operator[]()

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

Returns

Object at rhs.

H.24.5 Friends And Related Symbol Documentation

H.24.5.1 operator+

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

Returns

New iterator combining offsets.

H.24.5.2 operator-

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

Returns

New iterator differing offsets, iterating rhs' **AutoArray** (p. 306).

H.25 BiometricEvaluation::Memory::AutoBuffer< T > Class Template Reference

Public Types

- using **value_type** = T

  Manage a memory buffer.
- using **reference** = T&
- using **const_reference** = const T&

Public Member Functions

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

H.25.1 Member Typedef Documentation

H.25.1.1 value_type

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

Manage a memory buffer.

It's easier to think of **AutoBuffer** (p. 326) as a wrapper for a pointer rather than the object it truly is. Therefore, you can interact with the **AutoBuffer** (p. 326) 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** (p. 326) object like:

```
AutoBuffer<ANSI_NIST> obj = AutoBuffer(allocator_fn,
    deallocator_fn[, copy_constructor]);
```

Notice the **AutoBuffer** (p. 326) is for ANSI_NIST and not ANSI_NIST*, since **AutoBuffer** (p. 326) will handle the pointer for you. You can pass the **AutoBuffer**<**ANSI_NIST**> (p. 326) object to any function that takes an ANSI_NIST*. For example, it's perfectly valid to pass our 'obj' object above to:

```
write_fmttext(FILE *, ANSI_NIST *)
```

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

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

H.26 BiometricEvaluation::IO::AutoLogger Class Reference

The **AutoLogger** (p. 327) class provides an interface for writing to a log file within a background thread. The content for log entries is retrieved via a call back to the owning object.

`#include <be_io_autologger.h>`

Public Member Functions

- **AutoLogger** (**AutoLogger** const &)=delete
- **AutoLogger** & **operator=** (**AutoLogger** const &)=delete
- **AutoLogger** (**AutoLogger** &&)
- **AutoLogger** & **operator=** (**AutoLogger** &&)
- **AutoLogger** ()
- **AutoLogger** (const std::shared_ptr< **IO::Logsheet** > logSheet, const std::function< std::string()> &callback)

  *Construct an **AutoLogger** (p. 327) object that logs to an existing **Logsheet** (p. 585).*
- std::string **getComment** () const
- void **setComment** (std::string_view comment)

  Set a comment for each log entry.
- void **addLogEntry** ()

  *Create a log entry in the in the **Logsheet** (p. 585).*
- void **startAutoLogging** (std::chrono::microseconds interval)

  Start logging automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.
- void **stopAutoLogging** ()

  Stop automatic logging.
- pid_t **getTaskID** ()

  Return the task ID associated with this object.

H.26.1 Detailed Description

The **AutoLogger** (p. 327) class provides an interface for writing to a log file within a background thread. The content for log entries is retrieved via a call back to the owning object.

Auto logging will not start upon construction.

See also

startAutoLogging() (p. 329).

H.26.2 Constructor & Destructor Documentation

H.26.2.1 AutoLogger() [1/2]

```
BiometricEvaluation::IO::AutoLogger::AutoLogger ()
```

Constructor with no parameters.

H.26.2.2 AutoLogger() [2/2]

```
BiometricEvaluation::IO::AutoLogger::AutoLogger (
            const std::shared_ptr< IO::Logsheet > logSheet,
            const std::function< std::string()> & callback)
```

Construct an **AutoLogger** (p. 327) object that logs to an existing **Logsheet** (p. 585).

Parameters

in	*logSheet*	Existing **Logsheet** (p. 585) that will be appended.

H.26.3 Member Function Documentation

H.26.3.1 addLogEntry()

```
void BiometricEvaluation::IO::AutoLogger::addLogEntry ()
```

Create a log entry in the in the **Logsheet** (p. 585).

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when writing to the **Logsheet** (p. 585).

H.26.3.2 getComment()

```
std::string BiometricEvaluation::IO::AutoLogger::getComment () const
```

Returns

The comment string that is appended to each log entry.

H.26.3.3 getTaskID()

```
pid_t BiometricEvaluation::IO::AutoLogger::getTaskID ()
```

Return the task ID associated with this object.

The task ID is as seen by the OS and not any given threading library.

Returns

The task ID

H.26.3.4 setComment()

```
void BiometricEvaluation::IO::AutoLogger::setComment (
            std::string_view comment)
```

Set a comment for each log entry.

The comment string is auto-appended to the end of each log entry.

Parameters

comment	Comment string

H.26.3.5 startAutoLogging()

```
void BiometricEvaluation::IO::AutoLogger::startAutoLogging (
            std::chrono::microseconds interval)
```

Start logging 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()** (p. 329) is called very soon after the start, a log entry may not be made.

An interval value of 0 will not start auto-logging.

Parameters

in	*interval*	The gap between log entries, in microseconds.

Exceptions

Error::ObjectExists (p. 637)	Autologging is currently invoked.
Error::NotImplemented (p. 636)	The logging capability is not implemented for this operating system.

H.26.3.6 stopAutoLogging()

```
void BiometricEvaluation::IO::AutoLogger::stopAutoLogging ()
```

Stop automatic logging.

Exceptions

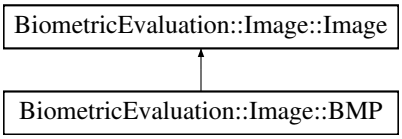
Error::ObjectDoesNotExist (p. 637)	Not currently autologging.
Error::StrategyError (p. 789)	An error occurred when stopping, most likely because the logging thread died.

H.27 BiometricEvaluation::Image::BMP Class Reference

A BMP-encoded image.

`#include <be_image_bmp.h>`

Inheritance diagram for BiometricEvaluation::Image::BMP:



Classes

- struct **ColorTableEntry**

Public Types

- using **ColorTableEntry** = struct ColorTableEntry
- using **ColorTable** = std::vector< **ColorTableEntry** >

Public Types inherited from BiometricEvaluation::Image::Image

- using **statusCallback_t**

Public Member Functions

- **BMP** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **BMP** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Memory::AutoArray**< uint8_t > **getRawData** () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::AutoArray**< uint8_t > **getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.

Public Member Functions inherited from BiometricEvaluation::Image::Image

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm** **getCompressionAlgorithm** () const

  Accessor for the CompressionAlgorithm of the image.
- **Resolution** **getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array** **getData** () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual **Memory::uint8Array** **getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Size** **getDimensions** () const

Accessor for the dimensions of the image in pixels.

- uint32_t **getColorDepth** () const

  Accessor for the color depth of the image in bits.

- uint16_t **getBitDepth** () const

  Accessor for the number of bits per color component.

- bool **hasAlphaChannel** () const

  Accessor for the presence of an alpha channel.

- statusCallback_t **getStatusCallback** () const

  Get handle to status callback function.

- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Static Public Member Functions

- static bool **isBMP** (const uint8_t *data, uint64_t size)

Static Public Member Functions inherited from BiometricEvaluation::Image::Image

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.

- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*

- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*

- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of an image file and create an **Image** (p. 477) object.*

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.

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

  *Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).*

- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Additional Inherited Members

Protected Member Functions inherited from BiometricEvaluation::Image::Image

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .
- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.
- void **setColorDepth** (const uint32_t colorDepth)

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

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

  Mutator for the presence of an alpha channel.

H.27.1 Detailed Description

A BMP-encoded image.

Note

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

H.27.2 Member Function Documentation

H.27.2.1 getRawData()

```
Memory::AutoArray< uint8_t > BiometricEvaluation::Image::BMP::getRawData () const  [virtual]
```

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

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 486).

H.27.2.2 getRawGrayscaleData()

```
Memory::AutoArray< uint8_t > BiometricEvaluation::Image::BMP::getRawGrayscaleData (
            uint8_t depth) const  [virtual]
```

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError (p. 390)	**Error** (p. 112) decompressing image data.
Error::NotImplemented (p. 636)	Unsupported conversion based on source color depth.
Error::ParameterError (p. 655)	Invalid value for depth.

Note

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

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implements **BiometricEvaluation::Image::Image** (p. 487).

H.27.2.3 isBMP()

```
static bool BiometricEvaluation::Image::BMP::isBMP (
            const uint8_t * data,
            uint64_t size)  [static]
```

Whether or not data is a **BMP** (p. 329) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a **BMP** (p. 329) image, false otherwise.

H.28 BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet Struct Reference

Public Member Functions

- **CharacterSet** (uint16_t **identifier**=0, std::string **commonName**="", std::string **version**="")
  *Create a new **CharacterSet** (p. 334) struct.*

Public Attributes

- uint16_t **identifier**
- std::string **commonName**
- std::string **version**

H.28.1 Constructor & Destructor Documentation

H.28.1.1 CharacterSet()

```
BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::CharacterSet (
            uint16_t identifier = 0,
            std::string commonName = "",
            std::string version = "")  [inline]
```

Create a new **CharacterSet** (p. 334) struct.

Parameters

identifier	Numeric identifier of the character set.

Parameters

commonName	Common name of the character set.
version	Optional version number of the character set.

H.28.2 Member Data Documentation

H.28.2.1 commonName

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

Common name of the character set

H.28.2.2 identifier

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

Identifier (000-999)

H.28.2.3 version

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

Optional version of the character set

H.29 BiometricEvaluation::Image::TIFF::ClientIO Struct Reference

`#include <be_image_tiff.h>`

Public Attributes

- **Memory::IndexedBuffer** * **ib** {nullptr}
- const **TIFF** * **tiffObject** {nullptr}

H.29.1 Detailed Description

Struct passed to libtiff client functions

H.29.2 Member Data Documentation

H.29.2.1 ib

`Memory::IndexedBuffer* BiometricEvaluation::Image::TIFF::ClientIO::ib {nullptr}`

Indexed buffer to **TIFF** (p. 804) object in memory.

H.29.2.2 tiffObject

`const TIFF* BiometricEvaluation::Image::TIFF::ClientIO::tiffObject {nullptr}`

Pointer to "this" **TIFF** (p. 804) object

H.30 BiometricEvaluation::Image::BMP::ColorTableEntry Struct Reference

`#include <be_image_bmp.h>`

Public Attributes

- uint8_t **red**
- uint8_t **green**
- uint8_t **blue**
- uint8_t **reserved**

H.30.1 Detailed Description

One element of the colormap table.

H.30.2 Member Data Documentation

H.30.2.1 blue

`uint8_t BiometricEvaluation::Image::BMP::ColorTableEntry::blue`

Blue value

H.30.2.2 green

`uint8_t BiometricEvaluation::Image::BMP::ColorTableEntry::green`

Green value

H.30.2.3 red

`uint8_t BiometricEvaluation::Image::BMP::ColorTableEntry::red`

Red value

H.30.2.4 reserved

`uint8_t BiometricEvaluation::Image::BMP::ColorTableEntry::reserved`

Reserved value

H.31 BiometricEvaluation::Process::CommandCenter< T, typename >::Command Class Reference

`#include <be_process_commandcenter.h>`

Public Attributes

- uint32_t **clientID**
- T **command**
- std::vector< std::string > **arguments**

H.31.1 Detailed Description

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>>
class BiometricEvaluation::Process::CommandCenter< T, typename >::Command

Parsed command received from the network.

H.31.2 Member Data Documentation

H.31.2.1 arguments

`template<typename T , typename = typename std::enable_if<std::is_enum<T>::value>>`
`std::vector<std::string>` **BiometricEvaluation::Process::CommandCenter**`< T, typename >::Command::arguments`

Arguments passed to command (optional).

H.31.2.2 clientID

`template<typename T , typename = typename std::enable_if<std::is_enum<T>::value>>`
`uint32_t` **BiometricEvaluation::Process::CommandCenter**`< T, typename >::Command::clientID`

ID of the sender.

H.31.2.3 command

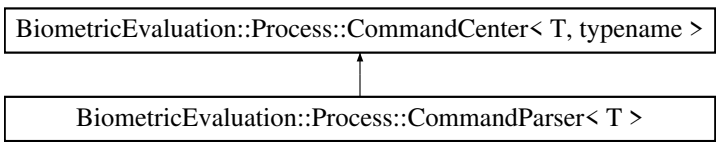
`template<typename T , typename = typename std::enable_if<std::is_enum<T>::value>>`
`T` **BiometricEvaluation::Process::CommandCenter**`< T, typename >::Command::command`

Enumeration value of the command.

H.32 BiometricEvaluation::Process::CommandCenter< T, typename > Class Template Reference

`#include <be_process_commandcenter.h>`

Inheritance diagram for BiometricEvaluation::Process::CommandCenter< T, typename >:



Classes

- class **Command**

Public Member Functions

- **CommandCenter** (uint16_t port= **MessageCenter::DEFAULT_PORT**)

  Constructor.
- ~**CommandCenter** ()=default
- bool **hasPendingCommands** ()

  Determine if there are commands waiting.
- bool **getNextCommand** (**Command** &command, int numSeconds=-1, std::string invalidCommand↩ Response="")

  Get the next command.
- void **sendResponse** (uint32_t clientID, const std::string &response, const std::string prefix=">> ", const std::string suffix="\n")

  Send a string response to a client.
- void **disconnectClient** (uint32_t clientID)

  Break the connection with a client.

H.32.1 Detailed Description

template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>>
class BiometricEvaluation::Process::CommandCenter< T, typename >

Receive enumerations as commands over the network.

H.32.2 Constructor & Destructor Documentation

H.32.2.1 CommandCenter()

```
template<typename T , typename = typename std::enable_if<std::is_enum<T>::value>>
BiometricEvaluation::Process::CommandCenter< T, typename >::CommandCenter (
            uint16_t port =  MessageCenter::DEFAULT_PORT)  [inline]
```

Constructor.

Parameters

port	Port to listen on for commands.

H.32.2.2 ~CommandCenter()

```
template<typename T , typename = typename std::enable_if<std::is_enum<T>::value>>
BiometricEvaluation::Process::CommandCenter< T, typename >::~ CommandCenter ()  [default]
```

Destructor (default).

H.32.3 Member Function Documentation

H.32.3.1 disconnectClient()

```
template<typename T , typename = typename std::enable_if<std::is_enum<T>::value>>
void  BiometricEvaluation::Process::CommandCenter< T, typename >::disconnectClient (
            uint32_t clientID)  [inline]
```

Break the connection with a client.

Parameters

clientID	ID of the client to disconect.

H.32.3.2 getNextCommand()

```
template<typename T , typename = typename std::enable_if<std::is_enum<T>::value>>
bool  BiometricEvaluation::Process::CommandCenter< T, typename >::getNextCommand (
             Command & command,
            int numSeconds = -1,
            std::string invalidCommandResponse = "")  [inline]
```

Get the next command.

Parameters

command	Reference to a **Command** (p. 337) that will be populated when this method returns true.

Parameters

numSeconds	Number of seconds to wait for a command, or -1 to block indefinitely.
invalidCommandResponse	Optional string to send, such as usage, that will be sent when an unrecognized command is received.

Returns

true if command has been populated, false otherwise.

H.32.3.3 hasPendingCommands()

```
template<typename T , typename = typename std::enable_if<std::is_enum<T>::value>>
bool BiometricEvaluation::Process::CommandCenter< T, typename >::hasPendingCommands () [inline]
```

Determine if there are commands waiting.

Returns

true if there are commands waiting, false otherwise.

Note

Returns immediately.

See also

BiometricEvaluation::Process::CommandCenter (p. 337):: **getNextCommand()** (p. 339)

H.32.3.4 sendResponse()

```
template<typename T , typename = typename std::enable_if<std::is_enum<T>::value>>
void  BiometricEvaluation::Process::CommandCenter< T, typename >::sendResponse (
            uint32_t clientID,
            const std::string & response,
            const std::string prefix = ">> ",
            const std::string suffix = "\n")  [inline]
```

Send a string response to a client.

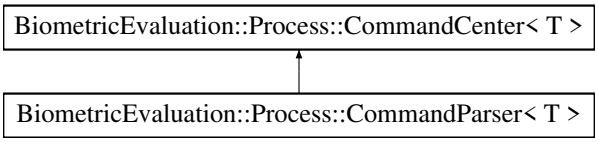
Parameters

clientID	ID of client to communicate with.
response	Printable string to send to client.
prefix	String to prefix to responses.
suffix	String to append to responses.

H.33 BiometricEvaluation::Process::CommandParser< T > Class Template Reference

`#include <be_process_commandcenter.h>`

Inheritance diagram for BiometricEvaluation::Process::CommandParser< T >:



Public Member Functions

- virtual void **parse** (const typename **CommandCenter**< T >::Command &command)=0

  Parse command.
- bool **getNextCommand** (typename **CommandCenter**< T >::Command &command, int numSeconds=-1)

  Get the next command.
- void **setUsage** (const std::string &usage)

  String sent when an invalid command is received.
- std::string **getUsage** () const
- **CommandParser** (uint16_t port= **MessageCenter::DEFAULT_PORT**)

  Constructor.
- virtual ~**CommandParser** ()=default

Public Member Functions inherited from BiometricEvaluation::Process::CommandCenter< T, typename >

- **CommandCenter** (uint16_t port= **MessageCenter::DEFAULT_PORT**)

  Constructor.
- ~**CommandCenter** ()=default
- bool **hasPendingCommands** ()

  Determine if there are commands waiting.
- bool **getNextCommand** (**Command** &command, int numSeconds=-1, std::string invalidCommand↩Response="")

  Get the next command.
- void **sendResponse** (uint32_t clientID, const std::string &response, const std::string prefix=">> ", const std::string suffix="\n")

  Send a string response to a client.
- void **disconnectClient** (uint32_t clientID)

  Break the connection with a client.

H.33.1 Detailed Description

template<typename T>
class BiometricEvaluation::Process::CommandParser< T >

Abstraction to parse messages received via **CommandCenter** (p. 337).

H.33.2 Constructor & Destructor Documentation

H.33.2.1 CommandParser()

```
template<typename T >
BiometricEvaluation::Process::CommandParser< T >::CommandParser (
            uint16_t port = MessageCenter::DEFAULT_PORT) [inline]
```

Constructor.

Parameters

port	Port to listen on for commands.

H.33.2.2 ~CommandParser()

```
template<typename T >
virtual BiometricEvaluation::Process::CommandParser< T >::~ CommandParser () [virtual], [default]
```

Virtual destructor (default).

H.33.3 Member Function Documentation

H.33.3.1 getNextCommand()

```
template<typename T >
bool BiometricEvaluation::Process::CommandParser< T >::getNextCommand (
            typename CommandCenter< T >::Command & command,
            int numSeconds = -1) [inline]
```

Get the next command.

Parameters

command	Reference to a Command that will be populated when this method returns true.

Parameters

numSeconds	Number of seconds to wait for a command, or -1 to block indefinitely.

Returns

true if command has been populated, false otherwise.

H.33.3.2 getUsage()

```
template<typename T >
std::string BiometricEvaluation::Process::CommandParser< T >::getUsage () const [inline]
```

Returns

Usage string.

H.33.3.3 parse()

```
template<typename T >
virtual void BiometricEvaluation::Process::CommandParser< T >::parse (
            const typename CommandCenter< T >::Command & command) [pure virtual]
```

Parse command.

Implement this method as a switch statement of your command enumeration.

H.33.3.4 setUsage()

```
template<typename T >
void BiometricEvaluation::Process::CommandParser< T >::setUsage (
            const std::string & usage) [inline]
```

String sent when an invalid command is received.

Parameters

usage	String to send when an invalid command is received.

Note

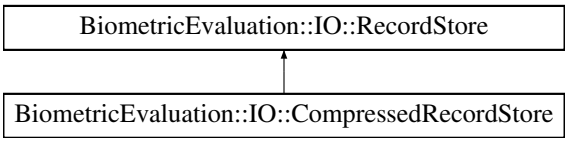
If not set, no additional usage is sent.

H.34 BiometricEvaluation::IO::CompressedRecordStore Class Reference

Sibling-implemented **IO::RecordStore** (p. 700) with Compression.

`#include <be_io_compressedrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::CompressedRecordStore:



Public Member Functions

- **CompressedRecordStore** (const std::string &pathname, const std::string &description, const **RecordStore::Kind** &recordStoreType, const std::string &compressorType)
- **CompressedRecordStore** (const std::string &pathname, const std::string &description, const **RecordStore::Kind** &recordStoreType, const **Compressor::Kind** &compressorType)
- **CompressedRecordStore** (const std::string &pathname, **IO::Mode** mode= **IO::Mode::ReadOnly**)
- uint64_t **getSpaceUsed** () const override

  Obtain real storage utilization.
- void **sync** () const override
- unsigned int **getCount** () const override
- std::string **getPathname** () const override
- std::string **getDescription** () const override
- void **changeDescription** (const std::string &description) override
- void **insert** (const std::string &key, const void *const data, const uint64_t size) override
- void **remove** (const std::string &key) override
- **Memory::uint8Array read** (const std::string &key) const override

  Read a complete record from a store.
- uint64_t **length** (const std::string &key) const override

- void **flush** (const std::string &key) const override
- **RecordStore::Record sequence** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  *Move the **RecordStore** (p. 700).*
- **CompressedRecordStore** (const **CompressedRecordStore** &rhs)=delete

  Copy constructor (disabled).
- **CompressedRecordStore** & **operator=** (const **CompressedRecordStore** &rhs)=delete

  Assignment operator (disabled).
- virtual void **insert** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const void *const data, const uint64_t size)

Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- virtual bool **containsKey** (const std::string &key) const

  *Determines whether the **RecordStore** (p. 700) contains an element with the specified key.*
- virtual **iterator begin** () noexcept
- virtual **iterator end** () noexcept

Additional Inherited Members

Public Types inherited from BiometricEvaluation::IO::RecordStore

- enum class **Kind** {
  BerkeleyDB , **Archive** , **File** , **SQLite** ,
  Compressed , **List** , **Default** = BerkeleyDB }
- using **Record** = struct Record
- using **iterator** = **IO::RecordStoreIterator**

Static Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- static bool **isRecordStore** (const std::string &pathname)

  *Determine if a location appears to be a **RecordStore** (p. 700).*
- static std::shared_ptr< **RecordStore** > **openRecordStore** (const std::string &pathname, **IO::Mode** mode= **Mode::ReadOnly**)

  *Open an existing **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static std::shared_ptr< **RecordStore** > **createRecordStore** (const std::string &pathname, const std::string &description, const **IO::RecordStore::Kind** &kind)

  *Create a new **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static void **removeRecordStore** (const std::string &pathname)
- static void **mergeRecordStores** (const std::string &mergePathname, const std::string &description, const **IO::RecordStore::Kind** &kind, const std::vector< std::string > &pathnames, const std::function< bool()> &interrupt=[ ]() {return(false);})

  *Create a new **RecordStore** (p. 700) that contains the contents of several other RecordStores.*

Static Public Attributes inherited from BiometricEvaluation::IO::RecordStore

- static const std::string **INVALIDKEYCHARS**
- static const int **BE_RECSTORE_SEQ_START** = 1
- static const int **BE_RECSTORE_SEQ_NEXT** = 2

H.34.1 Detailed Description

Sibling-implemented **IO::RecordStore** (p. 700) with Compression.

H.34.2 Constructor & Destructor Documentation

H.34.2.1 CompressedRecordStore() [1/4]

```
BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (
            const std::string & pathname,
            const std::string & description,
            const  RecordStore::Kind & recordStoreType,
            const std::string & compressorType)
```

Create a new **CompressedRecordStore** (p. 345), read/write mode.

Parameters

in	*pathname*	The directory where the store is to be created.
in	*description*	The store's description.
in	*recordStoreType*	The type of **RecordStore** (p. 700) subclass the internal RecordStores should be.

Parameters

in	*compressorType*	The type of compression that should be used within the internal RecordStores.

Exceptions

Error::ObjectExists (p. 637)	The store already exists.
Error::StrategyError (p. 789)	An error occurred when accessing the underlying file system.

H.34.2.2 CompressedRecordStore() [2/4]

```
BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (
            const std::string & pathname,
            const std::string & description,
            const  RecordStore::Kind & recordStoreType,
            const  Compressor::Kind & compressorType)
```

Create a new **CompressedRecordStore** (p. 345), read/write mode.

Parameters

in	*pathname*	The directory where the store is to be created.
in	*description*	The store's description.

Parameters

in	*recordStoreType*	The type of **RecordStore** (p. 700) subclass the internal RecordStores should be.
in	*compressorType*	The type of compression that should be used within the internal RecordStores.

Exceptions

Error::ObjectExists (p. 637)	The store already exists.
Error::StrategyError (p. 789)	An error occurred when accessing the underlying file system.

H.34.2.3 CompressedRecordStore() [3/4]

```
BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (
            const std::string & pathname,
             IO::Mode mode =  IO::Mode::ReadOnly)
```

Open an existing **CompressedRecordStore** (p. 345).

Parameters

in	*pathname*	The path name of the store.

Parameters

in	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The store does not exist.
Error::StrategyError (p. 789)	An error occurred when accessing the underlying file system.

H.34.2.4 CompressedRecordStore() [4/4]

```
BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (
            const CompressedRecordStore & rhs) [delete]
```

Copy constructor (disabled).

Disabled because this object could represent a file on disk.

Parameters

rhs	**CompressedRecordStore** (p. 345) object to copy.

H.34.3 Member Function Documentation

H.34.3.1 changeDescription()

```
void BiometricEvaluation::IO::CompressedRecordStore::changeDescription (
            const std::string & description) [override], [virtual]
```

Change the description of the **RecordStore** (p. 700).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 703).

H.34.3.2 flush()

```
void BiometricEvaluation::IO::CompressedRecordStore::flush (
            const std::string & key) const  [override], [virtual]
```

Commit the record's data to storage.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 704).

H.34.3.3 getCount()

```
unsigned int BiometricEvaluation::IO::CompressedRecordStore::getCount () const  [override], [virtual]
```

Obtain the number of items in the **RecordStore** (p. 700).

Returns

The number of items in the **RecordStore** (p. 700).

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.34.3.4 getDescription()

```
std::string BiometricEvaluation::IO::CompressedRecordStore::getDescription () const  [override],
[virtual]
```

Obtain a textual description of the **RecordStore** (p. 700).

Returns

The **RecordStore** (p. 700)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.34.3.5 getPathname()

```
std::string BiometricEvaluation::IO::CompressedRecordStore::getPathname () const  [override],
[virtual]
```

Return the path name of the **RecordStore** (p. 700).

Returns

Where in the file system the **RecordStore** (p. 700) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.34.3.6 getSpaceUsed()

```
uint64_t BiometricEvaluation::IO::CompressedRecordStore::getSpaceUsed () const  [override], [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** (p. 700).

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 706).

H.34.3.7 insert() [1/2]

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

Insert a record into the store.

Parameters

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

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 706).

H.34.3.8 insert() [2/2]

```
void BiometricEvaluation::IO::CompressedRecordStore::insert (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [override], [virtual]
```

Insert a record into the store.

Parameters

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

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Implements **BiometricEvaluation::IO::RecordStore** (p. 707).

H.34.3.9 length()

```
uint64_t BiometricEvaluation::IO::CompressedRecordStore::length (
            const std::string & key) const  [override], [virtual]
```

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 708).

H.34.3.10 move()

```
void BiometricEvaluation::IO::CompressedRecordStore::move (
            const std::string & pathname) [override], [virtual]
```

Move the **RecordStore** (p. 700).

The **RecordStore** (p. 700) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 700).

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 710).

H.34.3.11 operator=()

```
CompressedRecordStore & BiometricEvaluation::IO::CompressedRecordStore::operator= (
            const CompressedRecordStore & rhs) [delete]
```

Assignment operator (disabled).

Disabled because this object could represent a file on disk.

Parameters

rhs	**CompressedRecordStore** (p. 345) object to assign.

Returns

CompressedRecordStore (p. 345) object, now containing the contents of rhs.

H.34.3.12 read()

```
Memory::uint8Array BiometricEvaluation::IO::CompressedRecordStore::read (
            const std::string & key) const  [override], [virtual]
```

Read a complete record from a store.
The AutoArray will be resized to match the size of the data.

Parameters

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

Returns

The record associated with the key.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 712).

H.34.3.13 remove()

```
void BiometricEvaluation::IO::CompressedRecordStore::remove (
            const std::string & key)  [override], [virtual]
```

Remove a record from the store.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 713).

H.34.3.14 replace() [1/2]

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

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.34.3.15 replace() [2/2]

```
virtual void BiometricEvaluation::IO::RecordStore::replace (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [virtual]
```

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.34.3.16 sequence()

```
RecordStore::Record BiometricEvaluation::IO::CompressedRecordStore::sequence (
            int cursor = BE_RECSTORE_SEQ_NEXT)  [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the function to return the next record. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The record that is currently in sequence.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 715).

H.34.3.17 sequenceKey()

```
std::string BiometricEvaluation::IO::CompressedRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key.

Sequencing means to start at some point in the store and return the key, then repeatedly calling the function to return the next key. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The key of the currently sequenced record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 716).

H.34.3.18 setCursorAtKey()

```
void BiometricEvaluation::IO::CompressedRecordStore::setCursorAtKey (
            const std::string & key)  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 700), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 357).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 357).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.34.3.19 sync()

```
void BiometricEvaluation::IO::CompressedRecordStore::sync () const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

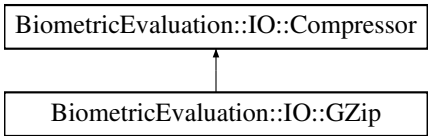
Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.35 BiometricEvaluation::IO::Compressor Class Reference

Common interface for classes providing compressing and decompressing functionality.

`#include <be_io_compressor.h>`

Inheritance diagram for BiometricEvaluation::IO::Compressor:



Public Types

- enum class **Kind** { **GZIP** }

Public Member Functions

- **Compressor** ()

  *Create a new **Compressor** (p. 360) object.*
- virtual **Memory::uint8Array** **compress** (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize) const =0

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

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

  Compress a buffer.
- virtual void **compress** (const **Memory::uint8Array** &uncompressedData, const std::string &outputFile) const =0

  Compress a buffer.
- virtual **Memory::uint8Array** **compress** (const std::string &inputFile) const =0

  Compress a file.
- virtual void **compress** (const std::string &inputFile, const std::string &outputFile) const =0

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

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

  Decompress a compressed buffer.
- virtual **Memory::uint8Array** **decompress** (const std::string &inputFile) const =0

Decompress a compressed buffer into a file.

- virtual void **decompress** (const **Memory::uint8Array** &compressedData, const std::string &output↩File) const =0

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

  Decompress a file.
- virtual void **decompress** (const std::string &inputFile, const std::string &outputFile) const =0

  Decompress a file.
- void **setOption** (const std::string &optionName, const std::string &optionValue)

  Assign a compressor option.
- void **setOption** (const std::string &optionName, int64_t optionValue)

  Assign a compressor option.
- std::string **getOption** (const std::string &optionName) const

  Obtain a compressor option as an integer.
- int64_t **getOptionAsInteger** (const std::string &optionName) const

  Obtain a compressor option as an integer.
- void **removeOption** (const std::string &optionName)

  Remove a compressor option.
- virtual ~**Compressor** ()
- **Compressor** (const **Compressor** &other)=delete

  Copy constructor (disabled).
- **Compressor** & **operator=** (const **Compressor** &other)=delete

  Assignment overload (disabled).

Static Public Member Functions

- static std::shared_ptr< **Compressor** > **createCompressor** (**Compressor::Kind** compressorKind=Kind↩::GZIP)

H.35.1 Detailed Description

Common interface for classes providing compressing and decompressing functionality.

H.35.2 Member Enumeration Documentation

H.35.2.1 Kind

`enum class` **`BiometricEvaluation::IO::Compressor::Kind`** `[strong]`

Kinds of Compressors (for factory)

H.35.3 Constructor & Destructor Documentation

H.35.3.1 Compressor() `[1/2]`

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

Create a new **Compressor** (p. 360) object.

Default compression options will be used.

H.35.3.2 ~Compressor()

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

Destructor

H.35.3.3 Compressor() [2/2]

```
BiometricEvaluation::IO::Compressor::Compressor (
            const  Compressor & other)  [delete]
```

Copy constructor (disabled).

Disabled because **Properties** (p. 674) member cannot be copied.

Parameters

other	**Compressor** (p. 360) to copy.

H.35.4 Member Function Documentation

H.35.4.1 compress() [1/6]

```
virtual  Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (
            const  Memory::uint8Array & uncompressedData) const  [pure virtual]
```

Compress a buffer.

Parameters

uncompressedData	Uncompressed data buffer to compress.

Returns

Compressed buffer.

Exceptions

Error::StrategyError (p. 789)	**Error** (p. 112) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 467).

H.35.4.2 compress() [2/6]

```
virtual void BiometricEvaluation::IO::Compressor::compress (
            const  Memory::uint8Array & uncompressedData,
            const std::string & outputFile) const  [pure virtual]
```

Compress a buffer.

Parameters

uncompressedData	Uncompressed data buffer to compress.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists (p. 637)	Output file already exists.
Error::StrategyError (p. 789)	**Error** (p. 112) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 467).

H.35.4.3 compress() [3/6]

```
virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::compress (
            const std::string & inputFile) const [pure virtual]
```

Compress a file.

Parameters

inputFile	Path to file to compress.

Returns

Compressed buffer.

Exceptions

Error::ObjectDoesNotExist (p. 637)	Input file does not exist.
Error::StrategyError (p. 789)	**Error** (p. 112) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 468).

H.35.4.4 compress() [4/6]

```
virtual void BiometricEvaluation::IO::Compressor::compress (
            const std::string & inputFile,
            const std::string & outputFile) const  [pure virtual]
```

Compress a file.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	Input file does not exist.
Error::ObjectExists (p. 637)	Output file already exists.
Error::StrategyError (p. 789)	**Error** (p. 112) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 469).

H.35.4.5 compress() [5/6]

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

Compress a buffer.

Parameters

uncompressedData	Uncompressed data buffer to compress.
uncompressedDataSize	Size of uncompressed↩ Data.

Returns

Compressed buffer.

Exceptions

Error::StrategyError (p. 789)	**Error** (p. 112) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 470).

H.35.4.6 compress() [6/6]

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

Compress a buffer.

Parameters

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

Exceptions

Error::ObjectExists (p. 637)	Output file already exists.
Error::StrategyError (p. 789)	**Error** (p. 112) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 471).

H.35.4.7 createCompressor()

```
static std::shared_ptr<  Compressor > BiometricEvaluation::IO::Compressor::createCompressor (
            Compressor::Kind compressorKind = Kind::GZIP)  [static]
```

Compressor (p. 360) factory.

Parameters

compressorKind	A known kind of compressor.

Returns

A new compressor with default options.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	Invalid compressor type.

H.35.4.8 decompress() [1/6]

```
virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (
            const Memory::uint8Array & compressedData) const [pure virtual]
```

Decompress a compressed buffer.

Parameters

compressedData	Compressed data buffer to decompress.

Returns

Decompressed data.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 471).

H.35.4.9 decompress() [2/6]

```
virtual void BiometricEvaluation::IO::Compressor::decompress (
            const Memory::uint8Array & compressedData,
            const std::string & outputFile) const [pure virtual]
```

Decompress a file.

Parameters

compressedData	Compressed data buffer to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists *(p. 637)*	Output file already exists.
Error::StrategyError *(p. 789)*	**Error** (p. 112) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 472).

H.35.4.10 decompress() [3/6]

```
virtual Memory::uint8Array BiometricEvaluation::IO::Compressor::decompress (
            const std::string & inputFile) const [pure virtual]
```

Decompress a compressed buffer into a file.

Parameters

inputFile	Location to save compressed file.

Returns

Decompressed data.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) in decompression unit.
Error::ObjectDoesNotExists	Output file already exists.

Implemented in **BiometricEvaluation::IO::GZip** (p. 473).

H.35.4.11 decompress() [4/6]

```
virtual void BiometricEvaluation::IO::Compressor::decompress (
            const std::string & inputFile,
            const std::string & outputFile) const  [pure virtual]
```

Decompress a file.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	Input file does not exist.
Error::ObjectExists (p. 637)	Output file already exists.
Error::StrategyError (p. 789)	**Error** (p. 112) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 473).

H.35.4.12 decompress() [5/6]

```
virtual void BiometricEvaluation::IO::Compressor::decompress (
            const uint8_t *const compressedData,
            const uint64_t compressedDataSize,
            const std::string & outputFile) const  [pure virtual]
```

Decompress a file.

Parameters

compressedData	Compressed data buffer to decompress.

Parameters

compressedDataSize	Size of compressed↩ Data.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists *(p. 637)*	Output file already exists.
Error::StrategyError *(p. 789)*	**Error** (p. 112) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 474).

H.35.4.13 decompress() [6/6]

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

Decompress a compressed buffer.

Parameters

compressedData	Compressed data buffer to decompress.
compressedDataSize	Size of compressed↩ Data.

Returns

Decompressed data.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 475).

H.35.4.14 getOption()

```
std::string BiometricEvaluation::IO::Compressor::getOption (
            const std::string & optionName) const
```

Obtain a compressor option as an integer.

Parameters

optionName	Name of the option to obtain.

Returns

Value of compressor option.

H.35.4.15 getOptionAsInteger()

```
int64_t BiometricEvaluation::IO::Compressor::getOptionAsInteger (
            const std::string & optionName) const
```

Obtain a compressor option as an integer.

Parameters

optionName	Name of the option to obtain.

Returns

Value of compressor option.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	The option was never set.

H.35.4.16 operator=()

```
Compressor & BiometricEvaluation::IO::Compressor::operator= (
            const  Compressor & other)  [delete]
```

Assignment overload (disabled).

Disabled because **Properties** (p. 674) member cannot be assigned.

Parameters

other	**Compressor** (p. 360) to assign.

Returns

lhs **Compressor** (p. 360).

H.35.4.17 removeOption()

```
void BiometricEvaluation::IO::Compressor::removeOption (
            const std::string & optionName)
```

Remove a compressor option.

Parameters

optionName	Name of the option to remove.

H.35.4.18 setOption() [1/2]

```
void BiometricEvaluation::IO::Compressor::setOption (
            const std::string & optionName,
            const std::string & optionValue)
```

Assign a compressor option.
Will overwrite existing values without warning.

Parameters

optionName	Name of the option to add.
optionValue	Value of the option.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) setting option.

H.35.4.19 setOption() [2/2]

```
void BiometricEvaluation::IO::Compressor::setOption (
            const std::string & optionName,
            int64_t optionValue)
```

Assign a compressor option.

Will overwrite existing values without warning.

Parameters

optionName	Name of the option to add.
optionValue	Value of the option.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) setting option.

H.36 BiometricEvaluation::Video::Container Class Reference

Representation of a video container.

`#include <be_video_container.h>`

Public Member Functions

- **Container** (const **Memory::uint8Array** &buffer)

  Construct a ***Container*** *(p. 374) from a memory buffer.*
- **Container** (const std::shared_ptr< **Memory::uint8Array** > &buffer)

  Construct a ***Container*** *(p. 374) from a memory buffer wrapped in a shared pointer.*
- **Container** (const std::string &filename)

  Construct a ***Container*** *(p. 374) from file.*
- uint32_t **getAudioCount** ()

  Obtain the number of audio streams.
- uint32_t **getVideoCount** ()

  Obtain the number of video streams.
- std::unique_ptr< **Video::Stream** > **getVideoStream** (uint32_t videoNum)

  Obtain a video stream from the container. ***Video*** *(p. 187) streams are indexed independently from other streams in the container.*

H.36.1 Detailed Description

Representation of a video container.

The **Container** (p. 374) class represents a single container stream that can be used to access the video and audio components of the stream.

H.36.2 Constructor & Destructor Documentation

H.36.2.1 Container() [1/3]

```
BiometricEvaluation::Video::Container::Container (
            const  Memory::uint8Array & buffer)
```

Construct a **Container** (p. 374) from a memory buffer.

Using this constructor can result in buffer memory usage twice that of other constructors.

Exceptions

Error::MemoryError *(p. 604)*	**Error** (p. 112) allocating memory for internal buffering.
Error::StrategyError *(p. 789)*	Other error when reading the container stream.

H.36.2.2 Container() [2/3]

```
BiometricEvaluation::Video::Container::Container (
            const std::shared_ptr<  Memory::uint8Array > & buffer)
```

Construct a **Container** (p. 374) from a memory buffer wrapped in a shared pointer.

Applications must not modify the data underlying the AutoArray.

Exceptions

Error::MemoryError *(p. 604)*	**Error** (p. 112) allocating memory for internal buffering.
Error::StrategyError *(p. 789)*	Other error when reading the container stream.

H.36.2.3 Container() [3/3]

```
BiometricEvaluation::Video::Container::Container (
            const std::string & filename)
```

Construct a **Container** (p. 374) from file.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	File does not exist.
Error::MemoryError *(p. 604)*	**Error** (p. 112) allocating memory for internal buffering.
Error::StrategyError *(p. 789)*	Other error when reading the container stream.

H.36.3 Member Function Documentation

H.36.3.1 getVideoStream()

```
std::unique_ptr<  Video::Stream > BiometricEvaluation::Video::Container::getVideoStream (
            uint32_t videoNum)
```

Obtain a video stream from the container. **Video** (p. 187) streams are indexed independently from other streams in the container.

Parameters

videoNum	The number of the video stream within the container.

Exceptions

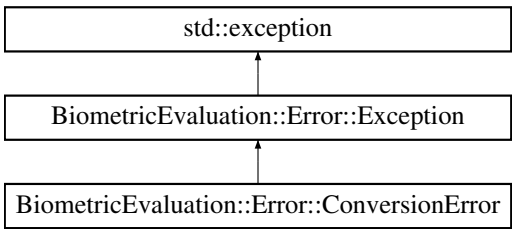
Error::ParameterError *(p. 655)*	The requested video stream is not available.

H.37 BiometricEvaluation::Error::ConversionError Class Reference

Error (p. 112) 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.37.1 Detailed Description

Error (p. 112) when converting one object into another, a property value from string to int, for example.

H.37.2 Constructor & Destructor Documentation

H.37.2.1 ConversionError() [1/2]

```
BiometricEvaluation::Error::ConversionError::ConversionError ()
```

Construct a **ConversionError** (p. 376) object with the default information string.

H.37.2.2 ConversionError() [2/2]

```
BiometricEvaluation::Error::ConversionError::ConversionError (
            const std::string & info)
```

Construct a **ConversionError** (p. 376) object with an information string appended to the default information string.

H.38 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** (p. 377) struct.*

Public Attributes

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

H.38.1 Detailed Description

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

H.38.2 Constructor & Destructor Documentation

H.38.2.1 Coordinate()

```
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** (p. 377) struct.

Parameters

in	*x*	X-coordinate
in	*y*	Y-coordinate

Parameters

in	*xDistance*	X-coordinate distance from origin
in	*yDistance*	Y-coordinate distance from origin

H.38.3 Member Data Documentation

H.38.3.1 x

`uint32_t BiometricEvaluation::Image::Coordinate::x`

X-coordinate

H.38.3.2 xDistance

`float BiometricEvaluation::Image::Coordinate::xDistance`

X-coordinate distance from origin

H.38.3.3 y

`uint32_t BiometricEvaluation::Image::Coordinate::y`

Y-coordinate

H.38.3.4 yDistance

`float BiometricEvaluation::Image::Coordinate::yDistance`

Y-coordinate distance from origin

H.39 BiometricEvaluation::Feature::AN2K11EFS::CorePoint Struct Reference

Public Attributes

- **Image::Coordinate location**
- bool **has_cdi**
- int **cdi**
- bool **has_rpu**
- int **rpu**
- bool **has_duy**
- int **duy**

H.40 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** (p. 379) struct.*

Public Attributes

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

H.40.1 Detailed Description

Representation of the core.

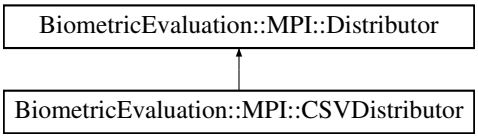
A core has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

H.41 BiometricEvaluation::MPI::CSVDistributor Class Reference

An implementation of the MPI::Distrbutor abstraction that distribute lines of a text file via work packages.

`#include <be_mpi_csvdistributor.h>`

Inheritance diagram for BiometricEvaluation::MPI::CSVDistributor:



Public Member Functions

- **CSVDistributor** (const std::string &propertiesFileName, const std::string &delimiter="")

  *Construct a **CSVDistributor** (p. 379) using named properties.*

Public Member Functions inherited from BiometricEvaluation::MPI::Distributor

- **Distributor** (const std::string &propertiesFileName)

  Constructor with properties file name.
- void **start** ()

  *Start of **MPI** (p. 162) processing for the distributor.*

Static Public Attributes

- static const std::string **CHECKPOINTLINECOUNT**
- static const std::string **CHECKPOINTRANDOMSEED**

Static Public Attributes inherited from BiometricEvaluation::MPI::Distributor

- static const std::string **CHECKPOINTFILENAME**
- static const std::string **CHECKPOINTREASON**
- static const std::string **CHECKPOINTPID**

Protected Member Functions

- void **createWorkPackage** (**MPI::WorkPackage** &workPackage)

  Create a work package for distribution.
- void **checkpointSave** (const std::string &reason)

  Create a checkpoint state.
- void **checkpointRestore** ()

  Restore from a checkpoint state.

Protected Member Functions inherited from BiometricEvaluation::MPI::Distributor

- std::shared_ptr< **IO::Logsheet** > **getLogsheet** () const

  Get access to the Logsheet object.
- std::shared_ptr< **IO::PropertiesFile** > **getCheckpointData** () const

  Get access to the checkpoint data object.

H.41.1 Detailed Description

An implementation of the MPI::Distrbutor abstraction that distribute lines of a text file via work packages.

This class supports checkpointing when an early exit is requested, allowing all workers to complete their current work package. If the input data lines were randomized, the random number generator seed is saved as part of the checkpoint.

On checkpoint restart, if the input data lines are randomized, the seed in the checkpoint must match the current seed; else an exception is thrown. If the checkpoint contains a seed, and the input is not currently randomized, and exception is thrown. See **MPI::CSVResources** (p. 386).

H.41.2 Constructor & Destructor Documentation

H.41.2.1 CSVDistributor()

```
BiometricEvaluation::MPI::CSVDistributor::CSVDistributor (
            const std::string & propertiesFileName,
            const std::string & delimiter = "")
```

Construct a **CSVDistributor** (p. 379) using named properties.

Parameters

in	*propertiesFileName*	The file containing the properties.

Parameters

in	*delimiter*	Delimiter used to tokenize lines read from CSV.

H.41.3 Member Function Documentation

H.41.3.1 checkpointRestore()

```
void BiometricEvaluation::MPI::CSVDistributor::checkpointRestore ()  [protected], [virtual]
```

Restore from a checkpoint state.

Implementations of this class use a checkpoint state to move the data sequence cursor to a point past data that has been previously distributed. The **MPI** (p. 162) **Framework** (p. 124) calls this method prior to the start of distributing work packages.

Implements **BiometricEvaluation::MPI::Distributor** (p. 406).

H.41.3.2 checkpointSave()

```
void BiometricEvaluation::MPI::CSVDistributor::checkpointSave (
            const std::string & reason)  [protected], [virtual]
```

Create a checkpoint state.

Implementations of this class create a checkpoint state that captures enough information to allow the implementation to move the data sequence cursor to a point past data that has been previously distributed. The **MPI** (p. 162) **Framework** (p. 124) calls this method when a premature shutdown is requested.

Parameters

reason	A string giving the reason for the checkpoint to be saved.

Implements **BiometricEvaluation::MPI::Distributor** (p. 407).

H.41.3.3 createWorkPackage()

```
void BiometricEvaluation::MPI::CSVDistributor::createWorkPackage (
            MPI::WorkPackage & workPackage)  [protected], [virtual]
```

Create a work package for distribution.

Implementations of this class create a work package to encapsulate the specific data type that is to be distributed.

Implements **BiometricEvaluation::MPI::Distributor** (p. 407).

H.41.4 Member Data Documentation

H.41.4.1 CHECKPOINTLINECOUNT

```
const std::string BiometricEvaluation::MPI::CSVDistributor::CHECKPOINTLINECOUNT [static]
```

The number of lines that were distributed, "Line Count".

H.41.4.2 CHECKPOINTRANDOMSEED

```
const std::string BiometricEvaluation::MPI::CSVDistributor::CHECKPOINTRANDOMSEED [static]
```

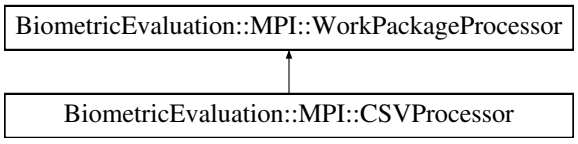
The seed used to randomize the input CSV file lines, "Random Seed".

H.42 BiometricEvaluation::MPI::CSVProcessor Class Reference

An implementation of a work package processor that will extract lines (and optionally tokenize) a line from a CSV text file.

`#include <be_mpi_csvprocessor.h>`

Inheritance diagram for BiometricEvaluation::MPI::CSVProcessor:



Public Member Functions

- **CSVProcessor** (const std::string &propertiesFileName)

  Construct a work package processor with the given properties.
- virtual void **processLine** (const uint64_t lineNum, const std::string &line)=0

  Method implemented by child classes to perform an action using each record from the Record Store.
- virtual std::shared_ptr< **WorkPackageProcessor** > **newProcessor** (std::shared_ptr< **IO::Logsheet** > &logsheet)=0

  Obtain an object that will process work packages. This method is part of the factory personality.
- virtual void **performInitialization** (std::shared_ptr< **IO::Logsheet** > &logsheet)=0

  Initialization function to be called before work is distributed to the work package processor.
- void **processWorkPackage** (**MPI::WorkPackage** &workPackage)

  Process (p. 170) *the data contents of the work package. This method is part of the worker personality.*

Public Member Functions inherited from BiometricEvaluation::MPI::WorkPackageProcessor

- virtual void **performShutdown** ()

  Terminiation function to be called during shut down after all work package processing is done.
- void **setLogsheet** (std::shared_ptr< **IO::Logsheet** > &logsheet)

*Set the **IO::Logsheet** (p. 585) object that can be used to save message for objects of this class.*

- std::shared_ptr< **IO::Logsheet** > **getLogsheet** ()

  *Obtain the **IO::Logsheet** (p. 585) object that can be used to save message for objects of this class.*

Protected Member Functions

- std::shared_ptr< **MPI::CSVResources** > **getResources** ()

H.42.1 Detailed Description

An implementation of a work package processor that will extract lines (and optionally tokenize) a line from a CSV text file.

Subclasses of this abstract class must implement the method to process the lines.

H.42.2 Constructor & Destructor Documentation

H.42.2.1 CSVProcessor()

```
BiometricEvaluation::MPI::CSVProcessor::CSVProcessor (
            const std::string & propertiesFileName)
```

Construct a work package processor with the given properties.

A **CSVProcessor** (p. 382) uses a text file to retrieve the data to be processed.

Note

Subclasses of this class should not manually read lines from the CSV.

The size of a single value item is limited to 2^64 octets. If the size of the value item is larger, behavior is undefined.

Parameters

in	*propertiesFileName*	The name of the file containing the properties for this object.

Exceptions

Error::Exception (p. 412)	An error occurred, usually due to missing or incorrect properties.

H.42.3 Member Function Documentation

H.42.3.1 newProcessor()

```
virtual std::shared_ptr< WorkPackageProcessor > BiometricEvaluation::MPI::CSVProcessor::newProcessor (
            std::shared_ptr< IO::Logsheet > & logsheet) [pure virtual]
```

Obtain an object that will process work packages. This method is part of the factory personality.

Parameters

logsheet	A shared pointer to the **IO::Logsheet** (p. 585) that may be used to save messages generated by the object.

Returns

A shared pointer to the work package processor.

Note

This method should always create a non-null **WorkPackageProcessor** (p. 843). If an error occurs during construction, throw a **Error::Exception** (p. 412) with a message to be caught and logged.

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 844).

H.42.3.2 performInitialization()

```
virtual void BiometricEvaluation::MPI::CSVProcessor::performInitialization (
            std::shared_ptr< IO::Logsheet > & logsheet) [pure virtual]
```

Initialization function to be called before work is distributed to the work package processor.

Implementations of this class can use this function to do any processing necessary before work is given to the processor, pre-forking.

This method is part of the factory personality. All state that is to be common across all package processor objects can be initialized in this method.

Parameters

logsheet	A shared pointer to the **IO::Logsheet** (p. 585) that may be used to save messages generated by the object.

Exceptions

Error::Exception *(p. 412)*	An implementation specific error occurred. The exception string will be logged by the **Framework** (

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 845).

H.42.3.3 processLine()

```
virtual void BiometricEvaluation::MPI::CSVProcessor::processLine (
            const uint64_t lineNum,
            const std::string & line)  [pure virtual]
```

Method implemented by child classes to perform an action using each record from the Record Store.

The source RecordStore must be accessible to the the implementation as the value for each key is not included.

Parameters

in	*lineNum*	The line number from the input file (1-based).

Parameters

in	*line*	The key associated with the record that is to be processed.

Exceptions

Error::Exception *(p. 412)*	An error occurred processing the record: Missing record, input/output error, or memory allocation.

H.42.3.4 processWorkPackage()

```
void BiometricEvaluation::MPI::CSVProcessor::processWorkPackage (
                MPI::WorkPackage & workPackage)  [virtual]
```

Process (p. 170) the data contents of the work package. This method is part of the worker personality.

Parameters

in	*workPackage*	The work package.

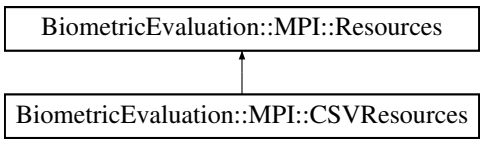
Exceptions

Error::Exception *(p. 412)*	An fatal error occurred when processing the work package; the processing responsible for this object

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 846).

H.43 BiometricEvaluation::MPI::CSVResources Class Reference

Inheritance diagram for BiometricEvaluation::MPI::CSVResources:



Public Member Functions

- **CSVResources** (const std::string &propertiesFileName)
- uint32_t **getChunkSize** () const
- bool **useBuffer** () const

  Obtain whether or not the entire CSV was read into memory at construction.
- bool **randomizeLines** () const

  If using buffer, whether or not to randomize how lines from the buffer are iterated.
- uint64_t **getNumRemainingLines** () const

  *Obtain the number of lines that have not yet been read from **readLine()** (p. 389) by a **Distributor** (p. 405).*
- std::string **getDelimiter** () const
- std::pair< uint64_t, std::string > **readLine** ()

  Obtain the next line from a buffer of file stream.
- uint64_t **getNumLines** () const

  Obtain number of lines of input.
- std::mt19937_64::result_type **getRandomSeed** () const

  Obtain the seed used to shuffle lines.

Public Member Functions inherited from BiometricEvaluation::MPI::Resources

- **Resources** (const std::string &propertiesFileName)

  Constructor taking the name of the properties file describing the resources.
- std::string **getPropertiesFileName** () const

  Obtain the name of the file used to construct this object.
- std::string **getLogsheetURL** () const

  *Obtain the Uniform Resource Locator for the **IO** (p. 136):Logsheet object.*
- std::string **getCheckpointPath** () const

  Obtain the Checkpoint Path name.
- int **getRank** () const
- int **getNumTasks** () const
- int **getWorkersPerNode** () const

Static Public Member Functions

- static std::vector< std::string > **getRequiredProperties** ()
- static std::vector< std::string > **getOptionalProperties** ()

Static Public Member Functions inherited from BiometricEvaluation::MPI::Resources

- static std::vector< std::string > **getRequiredProperties** ()

  Obtain the list of required properties.
- static std::vector< std::string > **getOptionalProperties** ()

  Obtain the list of optional properties.

Static Public Attributes

- static const std::string **INPUTCSVPROPERTY**
- static const std::string **CHUNKSIZEPROPERTY**
- static const std::string **USEBUFFERPROPERTY**
- static const std::string **RANDOMIZEPROPERTY**
- static const std::string **RANDOMSEEDPROPERTY**
- static const std::string **DELIMITERPROPERTY**
- static const std::string **TRIMPROPERTY**

Static Public Attributes inherited from BiometricEvaluation::MPI::Resources

- static const std::string **WORKERSPERNODEPROPERTY**

  The property string "Workers Per Node"; required.
- static const std::string **NUMCPUS**

  The "Workers Per Node" setting "NUMCPUS".
- static const std::string **NUMCORES**

  The "Workers Per Node" setting "NUMCORES".
- static const std::string **NUMSOCKETS**

  The "Workers Per Node" setting "NUMSOCKETS".
- static const std::string **LOGSHEETURLPROPERTY**

  The property string "Logsheet URL"; optional.
- static const std::string **CHECKPOINTPATHPROPERTY**

  The property string "Checkpoint Path"; required when checkpointing is enabled, optional otherwise.

H.43.1 Member Function Documentation

H.43.1.1 getDelimiter()

```
std::string BiometricEvaluation::MPI::CSVResources::getDelimiter () const
```

Returns

Delimiter used to tokenize sent lines.

H.43.1.2 getNumLines()

```
uint64_t BiometricEvaluation::MPI::CSVResources::getNumLines () const
```

Obtain number of lines of input.

Returns

Number of lines of input to send.

Exceptions

Error::StrategyError (p. 789)	Neither CSV file open nor CSV buffer populated.

H.43.1.3 getNumRemainingLines()

`uint64_t BiometricEvaluation::MPI::CSVResources::getNumRemainingLines () const`

Obtain the number of lines that have not yet been read from **readLine()** (p. 389) by a **Distributor** (p. 405).

Returns

Number of lines that have not been distributed.

H.43.1.4 getRandomSeed()

`std::mt19937_64::result_type BiometricEvaluation::MPI::CSVResources::getRandomSeed () const`

Obtain the seed used to shuffle lines.

Returns

Seed used to shuffle lines.

Exceptions

Error::StrategyError *(p. 789)*	Lines not randomized.

H.43.1.5 randomizeLines()

`bool BiometricEvaluation::MPI::CSVResources::randomizeLines () const`

If using buffer, whether or not to randomize how lines from the buffer are iterated.

Returns

true if RANDOMIZEPROPERTY and USEBUFFERPROPERTY are true, false otherwise.

H.43.1.6 readLine()

`std::pair< uint64_t, std::string > BiometricEvaluation::MPI::CSVResources::readLine ()`

Obtain the next line from a buffer of file stream.

Note

If _randomizeLines is true, sequential calls to this method will not necessarily return sequential lines.

Returns

The next line from buffer or file stream and the line number in the file where the line is from.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) with the file stream.
Error::ObjectDoesNotExist *(p. 637)*	File stream or buffer is exhausted.

H.43.1.7 useBuffer()

`bool BiometricEvaluation::MPI::CSVResources::useBuffer () const`

Obtain whether or not the entire CSV was read into memory at construction.

Returns

true if the entire INPUTCSVPROPERTY was read into memory at construction, false if an ifstream is kept open.

H.43.2 Member Data Documentation

H.43.2.1 CHUNKSIZEPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::CHUNKSIZEPROPERTY [static]`

Number of lines sent in succession

H.43.2.2 DELIMITERPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::DELIMITERPROPERTY [static]`

Delimiter to tokenize sent lines

H.43.2.3 INPUTCSVPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::INPUTCSVPROPERTY [static]`

Text (p. 173) file to read

H.43.2.4 RANDOMIZEPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::RANDOMIZEPROPERTY [static]`

Randomly iterate buffer

H.43.2.5 RANDOMSEEDPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::RANDOMSEEDPROPERTY [static]`

Seed for randomization

H.43.2.6 TRIMPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::TRIMPROPERTY [static]`

Trim whitespace from lines read

H.43.2.7 USEBUFFERPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::USEBUFFERPROPERTY [static]`

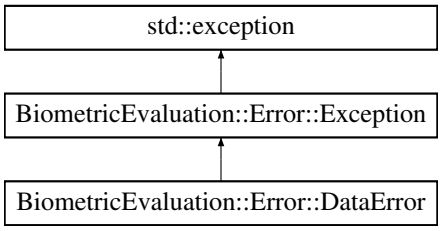
Read file into buffer first, or read from file

H.44 BiometricEvaluation::Error::DataError Class Reference

Error (p. 112) when reading data from an external source.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::DataError:



Public Member Functions

- **DataError** ()
- **DataError** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.44.1 Detailed Description

Error (p. 112) when reading data from an external source.

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

H.44.2 Constructor & Destructor Documentation

H.44.2.1 DataError() [1/2]

```
BiometricEvaluation::Error::DataError::DataError ()
```

Construct a **DataError** (p. 390) object with the default information string.

H.44.2.2 DataError() [2/2]

```
BiometricEvaluation::Error::DataError::DataError (
            const std::string & info)
```

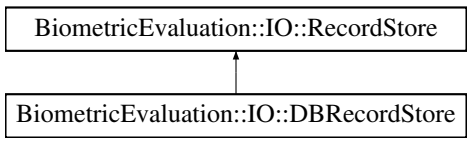
Construct a **DataError** (p. 390) object with an information string appended to the default information string.

H.45 BiometricEvaluation::IO::DBRecordStore Class Reference

A class that implements **IO::RecordStore** (p. 700) using a Berkeley DB database as the underlying record storage system.

`#include <be_io_dbrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::DBRecordStore:



Public Member Functions

- **DBRecordStore** (const std::string &pathname, const std::string &description)
- **DBRecordStore** (const std::string &pathname, **IO::Mode** mode= **IO::Mode::ReadOnly**)
- **Memory::uint8Array read** (const std::string &key) const override

  Read a complete record from a store.
- void **insert** (const std::string &key, const void *const data, const uint64_t size) override
- void **remove** (const std::string &key) override
- uint64_t **length** (const std::string &key) const override
- void **flush** (const std::string &key) const override
- **RecordStore::Record sequence** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  *Move the **RecordStore** (p. 700).*
- uint64_t **getSpaceUsed** () const override

  Obtain real storage utilization.
- void **sync** () const override
- unsigned int **getCount** () const override
- std::string **getPathname** () const override
- std::string **getDescription** () const override
- void **changeDescription** (const std::string &description) override
- **DBRecordStore** (const **DBRecordStore** &)=delete
- **DBRecordStore** & **operator=** (const **DBRecordStore** &)=delete
- virtual void **insert** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const void *const data, const uint64_t size)

Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- virtual bool **containsKey** (const std::string &key) const

  *Determines whether the **RecordStore** (p. 700) contains an element with the specified key.*
- virtual **iterator begin** () noexcept
- virtual **iterator end** () noexcept

Additional Inherited Members

Public Types inherited from BiometricEvaluation::IO::RecordStore

- enum class **Kind** {
  BerkeleyDB , **Archive** , **File** , **SQLite** ,
  Compressed , **List** , **Default** = BerkeleyDB }
- using **Record** = struct Record
- using **iterator** = **IO::RecordStoreIterator**

Static Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- static bool **isRecordStore** (const std::string &pathname)

  *Determine if a location appears to be a **RecordStore** (p. 700).*
- static std::shared_ptr< **RecordStore** > **openRecordStore** (const std::string &pathname, **IO::Mode** mode= **Mode::ReadOnly**)

  *Open an existing **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static std::shared_ptr< **RecordStore** > **createRecordStore** (const std::string &pathname, const std::string &description, const **IO::RecordStore::Kind** &kind)

  *Create a new **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static void **removeRecordStore** (const std::string &pathname)
- static void **mergeRecordStores** (const std::string &mergePathname, const std::string &description, const **IO::RecordStore::Kind** &kind, const std::vector< std::string > &pathnames, const std::function< bool()> &interrupt=[ ]() {return(false);})

  *Create a new **RecordStore** (p. 700) that contains the contents of several other RecordStores.*

Static Public Attributes inherited from BiometricEvaluation::IO::RecordStore

- static const std::string **INVALIDKEYCHARS**
- static const int **BE_RECSTORE_SEQ_START** = 1
- static const int **BE_RECSTORE_SEQ_NEXT** = 2

H.45.1 Detailed Description

A class that implements **IO::RecordStore** (p. 700) using a Berkeley DB database as the underlying record storage system.

H.45.2 Constructor & Destructor Documentation

H.45.2.1 DBRecordStore() [1/2]

```
BiometricEvaluation::IO::DBRecordStore::DBRecordStore (
            const std::string & pathname,
            const std::string & description)
```

Create a new **DBRecordStore** (p. 391), read/write mode.

Parameters

in	*pathname*	The directory where the store will be created.
in	*description*	The store's description.

Exceptions

Error::ObjectExists (p. 637)	The store already exists.
Error::StrategyError (p. 789)	An error occurred when accessing the underlying file system.

H.45.2.2 DBRecordStore() [2/2]

```
BiometricEvaluation::IO::DBRecordStore::DBRecordStore (
            const std::string & pathname,
             IO::Mode mode =  IO::Mode::ReadOnly)
```

Open an existing **DBRecordStore** (p. 391).

Parameters

in	*name*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The store does not exist.
Error::StrategyError (p. 789)	An error occurred when accessing the underlying file system.

H.45.3 Member Function Documentation

H.45.3.1 changeDescription()

```
void BiometricEvaluation::IO::DBRecordStore::changeDescription (
            const std::string & description)  [override], [virtual]
```

Change the description of the **RecordStore** (p. 700).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 703).

H.45.3.2 flush()

```
void BiometricEvaluation::IO::DBRecordStore::flush (
            const std::string & key) const  [override], [virtual]
```

Commit the record's data to storage.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 704).

H.45.3.3 getCount()

```
unsigned int BiometricEvaluation::IO::DBRecordStore::getCount () const  [override], [virtual]
```

Obtain the number of items in the **RecordStore** (p. 700).

Returns

The number of items in the **RecordStore** (p. 700).

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.45.3.4 getDescription()

```
std::string BiometricEvaluation::IO::DBRecordStore::getDescription () const  [override], [virtual]
```

Obtain a textual description of the **RecordStore** (p. 700).

Returns

The **RecordStore** (p. 700)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.45.3.5 getPathname()

```
std::string BiometricEvaluation::IO::DBRecordStore::getPathname () const  [override], [virtual]
```

Return the path name of the **RecordStore** (p. 700).

Returns

Where in the file system the **RecordStore** (p. 700) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.45.3.6 getSpaceUsed()

```
uint64_t BiometricEvaluation::IO::DBRecordStore::getSpaceUsed () const  [override], [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** (p. 700).

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 706).

H.45.3.7 insert() [1/2]

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

Insert a record into the store.

Parameters

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

Parameters

in	*data*	The data for the record.

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 706).

H.45.3.8 insert() [2/2]

```
void BiometricEvaluation::IO::DBRecordStore::insert (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [override], [virtual]
```

Insert a record into the store.

Parameters

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

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Implements **BiometricEvaluation::IO::RecordStore** (p. 707).

H.45.3.9 length()

```
uint64_t BiometricEvaluation::IO::DBRecordStore::length (
            const std::string & key) const  [override], [virtual]
```

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 708).

H.45.3.10 move()

```
void BiometricEvaluation::IO::DBRecordStore::move (
            const std::string & pathname)  [override], [virtual]
```

Move the **RecordStore** (p. 700).

The **RecordStore** (p. 700) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 700).

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 710).

H.45.3.11 read()

```
Memory::uint8Array BiometricEvaluation::IO::DBRecordStore::read (
            const std::string & key) const  [override], [virtual]
```

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

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

Returns

The record associated with the key.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 712).

H.45.3.12 remove()

```
void BiometricEvaluation::IO::DBRecordStore::remove (
            const std::string & key)  [override], [virtual]
```

Remove a record from the store.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 713).

H.45.3.13 replace() [1/2]

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

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.45.3.14 replace() [2/2]

```
virtual void BiometricEvaluation::IO::RecordStore::replace (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [virtual]
```

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Parameters

in	*size*	The size of the record, in bytes.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.45.3.15 sequence()

```
RecordStore::Record BiometricEvaluation::IO::DBRecordStore::sequence (
        int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the function to return the next record. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The record that is currently in sequence.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 715).

H.45.3.16 sequenceKey()

```
std::string BiometricEvaluation::IO::DBRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key.

Sequencing means to start at some point in the store and return the key, then repeatedly calling the function to return the next key. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The key of the currently sequenced record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 716).

H.45.3.17 setCursorAtKey()

```
void BiometricEvaluation::IO::DBRecordStore::setCursorAtKey (
            const std::string & key) [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 700), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 401).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 401).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.45.3.18 sync()

```
void BiometricEvaluation::IO::DBRecordStore::sync () const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.46 BiometricEvaluation::Feature::AN2K11EFS::DeltaPoint Struct Reference

Representation of an extended feature set delta.

```
#include <be_feature_an2k11efs.h>
```

Public Attributes

- **Image::Coordinate location**
- bool **has_dup**
- int **dup**

- bool **has_dlf**
- int **dlf**
- bool **has_drt**
- int **drt**
- bool **has_dtp**
- DeltaType **dtp**
- bool **has_rpu**
- int **rpu**
- bool **has_duu**
- int **duu**
- bool **has_dul**
- int **dul**
- bool **has_dur**
- int **dur**

H.46.1 Detailed Description

Representation of an extended feature set delta.

H.47 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** (p. 404) struct.*

Public Attributes

- **Image::Coordinate coordinate**
- bool **has_angle**
- int **angle1**
- int **angle2**
- int **angle3**

H.47.1 Detailed Description

Representation of the delta.

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

H.48 DIR Struct Reference

Public Attributes

- struct **dirent ent**
- struct **_WDIR** * **wdirp**

H.49 dirent Struct Reference

Public Attributes

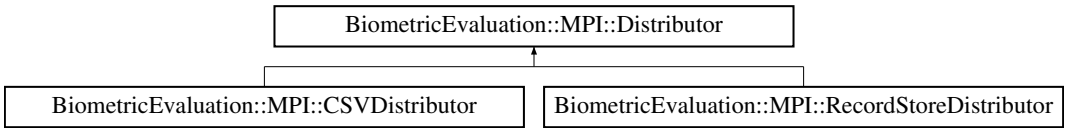
- long **d_ino**
- long **d_off**
- unsigned short **d_reclen**
- size_t **d_namlen**
- int **d_type**
- char **d_name** [PATH_MAX+1]

H.50 BiometricEvaluation::MPI::Distributor Class Reference

A class to represent an **MPI** (p. 162) task that distributes work to other tasks.

`#include <be_mpi_distributor.h>`

Inheritance diagram for BiometricEvaluation::MPI::Distributor:



Public Member Functions

- **Distributor** (const std::string &propertiesFileName)

  Constructor with properties file name.
- void **start** ()

  *Start of **MPI** (p. 162) processing for the distributor.*

Static Public Attributes

- static const std::string **CHECKPOINTFILENAME**
- static const std::string **CHECKPOINTREASON**
- static const std::string **CHECKPOINTPID**

Protected Member Functions

- virtual void **createWorkPackage** (**MPI::WorkPackage** &workPackage)=0

  Create a work package for distribution.
- virtual void **checkpointSave** (const std::string &reason)=0

  Create a checkpoint state.
- virtual void **checkpointRestore** ()=0

  Restore from a checkpoint state.
- std::shared_ptr< **IO::Logsheet** > **getLogsheet** () const

  Get access to the Logsheet object.
- std::shared_ptr< **IO::PropertiesFile** > **getCheckpointData** () const

  Get access to the checkpoint data object.

H.50.1 Detailed Description

A class to represent an **MPI** (p. 162) task that distributes work to other tasks.

A **Distributor** (p. 405) object is based on a set of properties contained in a file. This class must be subclassed and an implementation of the **createWorkPackage()** (p. 407) method provided.

The distributor sends an **MPI** (p. 162) message to each receiver object indicating whether it should start and ready for accepting work packages, or proceed immediately to the shutdown state. Failure to start the **Distributor** (p. 405) object will result in the entire **MPI** (p. 162) job shutting down before any work is done.

If the Logsheet URL property is set, log messages will be written to that sheet. Otherwise, log messages will be written to a Null Logsheet.

See also

IO::Properties (p. 674)

MPI::Receiver (p. 692)

MPI::WorkPackage (p. 841)

H.50.2 Constructor & Destructor Documentation

H.50.2.1 Distributor()

```
BiometricEvaluation::MPI::Distributor::Distributor (
            const std::string & propertiesFileName)
```

Constructor with properties file name.

Parameters

in	*propertiesFileName*	The name of the file containing the properties for the new object.

Exceptions

Error::Exception (p. 412)	An error occurred, possibly due to missing or invalid properties.

H.50.3 Member Function Documentation

H.50.3.1 checkpointRestore()

```
virtual void BiometricEvaluation::MPI::Distributor::checkpointRestore ()  [protected], [pure virtual]
```

Restore from a checkpoint state.

Implementations of this class use a checkpoint state to move the data sequence cursor to a point past data that has been previously distributed. The **MPI** (p. 162) **Framework** (p. 124) calls this method prior to the start of distributing work packages.

Implemented in **BiometricEvaluation::MPI::CSVDistributor** (p. 381), and **BiometricEvaluation::↩ MPI::RecordStoreDistributor** (p. 720).

H.50.3.2 checkpointSave()

```
virtual void BiometricEvaluation::MPI::Distributor::checkpointSave (
            const std::string & reason)  [protected], [pure virtual]
```

Create a checkpoint state.

Implementations of this class create a checkpoint state that captures enough information to allow the implementation to move the data sequence cursor to a point past data that has been previously distributed. The **MPI** (p. 162) **Framework** (p. 124) calls this method when a premature shutdown is requested.

Parameters

reason	A string giving the reason for the checkpoint to be saved.

Implemented in **BiometricEvaluation::MPI::CSVDistributor** (p. 381), and **BiometricEvaluation::↩ MPI::RecordStoreDistributor** (p. 720).

H.50.3.3 createWorkPackage()

```
virtual void BiometricEvaluation::MPI::Distributor::createWorkPackage (
             MPI::WorkPackage & workPackage)  [protected], [pure virtual]
```

Create a work package for distribution.

Implementations of this class create a work package to encapsulate the specific data type that is to be distributed.

Implemented in **BiometricEvaluation::MPI::CSVDistributor** (p. 381), and **BiometricEvaluation::↩ MPI::RecordStoreDistributor** (p. 721).

H.50.3.4 getCheckpointData()

```
std::shared_ptr< IO::PropertiesFile > BiometricEvaluation::MPI::Distributor::getCheckpoint↩
Data () const  [protected]
```

Get access to the checkpoint data object.

Returns

A shared pointer for the checkpoint data object.

H.50.3.5 getLogsheet()

`std::shared_ptr< IO::Logsheet > BiometricEvaluation::MPI::Distributor::getLogsheet () const [protected]`

Get access to the Logsheet object.

Returns

A shared pointer for the Logsheet object.

H.50.3.6 start()

`void BiometricEvaluation::MPI::Distributor::start ()`

Start of **MPI** (p. 162) processing for the distributor.

Once started, the distributor will send a message to each receiver task telling it to start and wait for status back from each receiver.

H.50.4 Member Data Documentation

H.50.4.1 CHECKPOINTFILENAME

`const std::string BiometricEvaluation::MPI::Distributor::CHECKPOINTFILENAME [static]`

The name of the checkpoint properties file, "Distributor.chk".

H.50.4.2 CHECKPOINTPID

`const std::string BiometricEvaluation::MPI::Distributor::CHECKPOINTPID [static]`

The process ID of the checkpointing **Distributor** (p. 405) process, "PID".

H.50.4.3 CHECKPOINTREASON

`const std::string BiometricEvaluation::MPI::Distributor::CHECKPOINTREASON [static]`

The reason string given for the checkpoint to be taken, "Reason".

H.51 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** (std::string **identifier**="", std::string **version**="")

  *Create a **DomainName** (p. 408) struct.*

Public Attributes

- std::string **identifier**
- std::string **version**

H.51.1 Detailed Description

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

H.51.2 Constructor & Destructor Documentation

H.51.2.1 DomainName()

```
BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::DomainName (
            std::string identifier = "",
            std::string version = "")  [inline]
```

Create a **DomainName** (p. 408) struct.

Parameters

identifier	Unique identifier for agency, entity, or implementation.
version	Optional unique version number of the implementation of the identifier.

H.51.3 Member Data Documentation

H.51.3.1 identifier

```
std::string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::identifier
```

Unique identifier for agency, entity, or implementation.

H.51.3.2 version

```
std::string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::version
```

Optional version of the implementation

H.52 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry Struct Reference

Public Member Functions

- **Entry** (bool **standard**, std::string **code**)

Public Attributes

- bool **standard**
- std::string **code**

H.52.1 Constructor & Destructor Documentation

H.52.1.1 Entry()

```
BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::Entry (
            bool standard,
            std::string code)
```

Create an **Entry** (p. 409) struct.

Parameters

standard	Whether or not code is a standard AN2K pattern classification code.
code	AN2K or user-defined pattern classification code.

H.52.2 Member Data Documentation

H.52.2.1 code

```
std::string BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::code
```

AN2K or user-defined pattern classification code.

H.52.2.2 standard

```
bool BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::standard
```

Whether code is a standard AN2K pattern classification code.

H.53 BiometricEvaluation::Feature::AN2K11EFS::Examiner↩ AnalysisAssessment Struct Reference

`#include <be_feature_an2k11efs.h>`

Public Attributes

- bool **present** {false}
- ValueAssessmentCode **aav**
- std::string **aln**
- std::string **afn**
- std::string **aaf**
- std::string **amt**
- std::string **acm** {}
- bool **has_cxf** {false}
- bool **cxf** {}

H.53.1 Detailed Description

Examiner's assessment of an impression

H.53.2 Member Data Documentation

H.53.2.1 aaf

`std::string BiometricEvaluation::Feature::AN2K11EFS::ExaminerAnalysisAssessment::aaf`
Examiner's employer/affiliation (required)

H.53.2.2 aav

`ValueAssessmentCode BiometricEvaluation::Feature::AN2K11EFS::ExaminerAnalysisAssessment::aav`
Value of impression (required)

H.53.2.3 acm

`std::string BiometricEvaluation::Feature::AN2K11EFS::ExaminerAnalysisAssessment::acm` {}
Comment (optional)

H.53.2.4 afn

`std::string BiometricEvaluation::Feature::AN2K11EFS::ExaminerAnalysisAssessment::afn`
Examiner's first and middle names (required)

H.53.2.5 aln

`std::string BiometricEvaluation::Feature::AN2K11EFS::ExaminerAnalysisAssessment::aln`
Examiner's surname (required)

H.53.2.6 amt

`std::string BiometricEvaluation::Feature::AN2K11EFS::ExaminerAnalysisAssessment::amt`
Date and time determination made (GMT, required)

H.53.2.7 cxf

`bool BiometricEvaluation::Feature::AN2K11EFS::ExaminerAnalysisAssessment::cxf {}`

Whether analysis was complex (optional)

H.53.2.8 has_cxf

`bool BiometricEvaluation::Feature::AN2K11EFS::ExaminerAnalysisAssessment::has_cxf {false}`

Whether `cxf` is populated (required)

H.53.2.9 present

`bool BiometricEvaluation::Feature::AN2K11EFS::ExaminerAnalysisAssessment::present {false}`

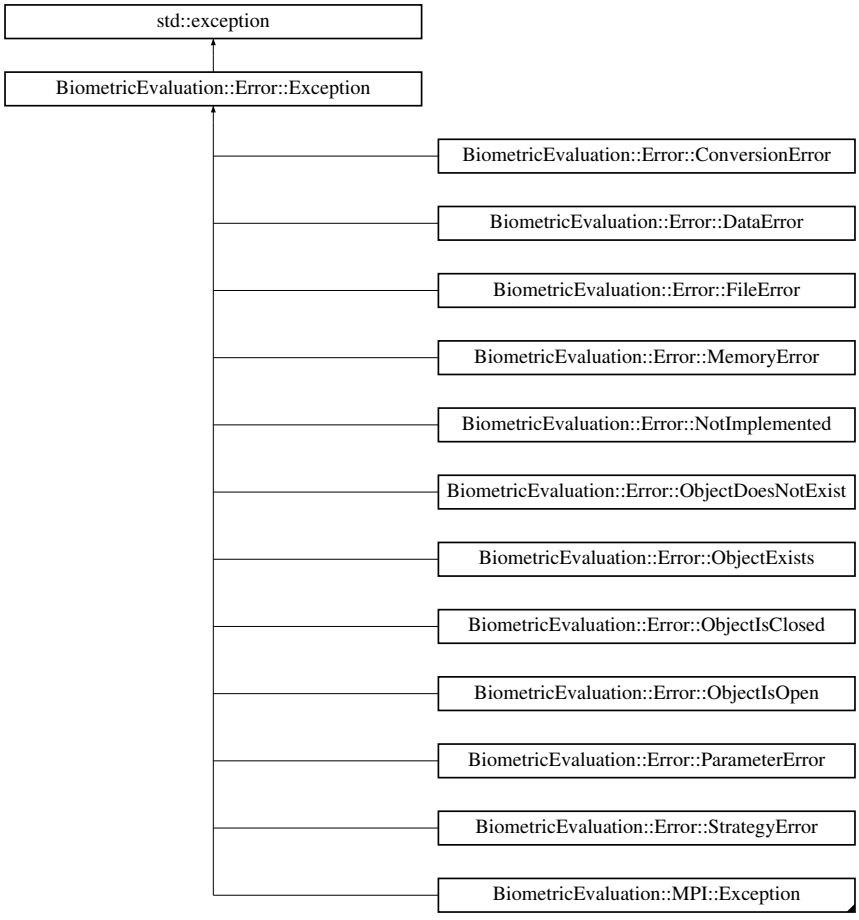
Whether this field was present

H.54 BiometricEvaluation::Error::Exception Class Reference

The parent class of all **BiometricEvaluation** (p. 111) exceptions.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::Exception:



Public Member Functions

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.54.1 Detailed Description

The parent class of all **BiometricEvaluation** (p. 111) exceptions.

The classes derived from this class will have a default information string set indicating the type of exception. Any additional information string is appended to that string.

H.54.2 Constructor & Destructor Documentation

H.54.2.1 Exception() [1/2]

```
BiometricEvaluation::Error::Exception::Exception ()
```

Construct an **Exception** (p. 412) object without an information string.

H.54.2.2 Exception() [2/2]

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

Construct an **Exception** (p. 412) object with an information string.

Parameters

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

H.54.2.3 ~Exception()

```
virtual BiometricEvaluation::Error::Exception::~Exception ()  [virtual], [default]
```

Reimplemented in **BiometricEvaluation::MPI::Exception** (p. 415).

H.54.3 Member Function Documentation

H.54.3.1 what()

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

Obtain the information string associated with the exception.

Returns

The information string as a char array.

H.54.3.2 whatString()

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

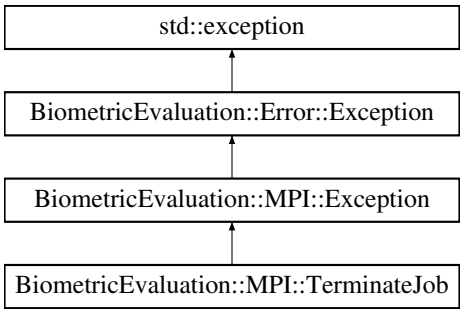
Obtain the information string associated with the exception.

Returns

The information string.

H.55 BiometricEvaluation::MPI::Exception Class Reference

Inheritance diagram for BiometricEvaluation::MPI::Exception:



Public Member Functions

- **Exception** ()
- **Exception** (std::string info)

  Constructor.
- virtual ~**Exception** () noexcept=default

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.55.1 Constructor & Destructor Documentation

H.55.1.1 Exception() [1/2]

```
BiometricEvaluation::MPI::Exception::Exception ()
```

Construct with default information string.

H.55.1.2 Exception() [2/2]

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

Constructor.

Parameters

info	Custom information string. Will be appended to the default information string.

H.55.1.3 ~Exception()

```
virtual BiometricEvaluation::MPI::Exception::~Exception ()  [virtual], [default], [noexcept]
```

Destructor.

Reimplemented from **BiometricEvaluation::Error::Exception** (p. 412).

H.56 BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeatureSet Class Reference

A class to represent the Extended **Feature** (p. 115) Set optionally present in an ANSI/NIST Type-9 record.

```
#include <be_feature_an2k11efs.h>
```

Public Member Functions

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

  Construct an AN2K11 EFS object from file data.
- **ExtendedFeatureSet** (**Memory::uint8Array** &buf, int recordNumber)

  Construct an AN2K11 EFS object from data contained in a memory buffer.
- **ImageInfo getImageInfo** () const

  *Obtain the structure containing information about the image and Extended **Feature** (p. 115) Set.*
- BiometricEvaluation::Feature::AN2K11EFS::MinutiaPointSet **getMPS** () const

  Obtain the minutiae point set.
- **BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCountInfo getMRCI** () const

  Obtain all the information relating to minutiae ridge count information.
- BiometricEvaluation::Feature::AN2K11EFS::CorePointSet **getCPS** () const

  Obtain the core point set.
- BiometricEvaluation::Feature::AN2K11EFS::DeltaPointSet **getDPS** () const

Obtain the delta point set.

- std::vector< LatentProcessingMethod > **getLPM** () const

  Obtain set of methods used to process this latent.
- **BiometricEvaluation::Feature::AN2K11EFS::NoFeaturesPresent getNFP** () const
- **ExaminerAnalysisAssessment getEAA** () const

  Obtain the examiner's analysis assessment of the print.
- **Substrate getLSB** () const
- std::vector< **Pattern** > **getPAT** () const

H.56.1 Detailed Description

A class to represent the Extended **Feature** (p. 115) Set optionally present in an ANSI/NIST Type-9 record.

Each minutiae point, ridge count item, core, and delta is represented in the native ANSI/NIST format. Conforms with ANSI/NIST-ITL-2011: Update 2015 standard.

H.56.2 Constructor & Destructor Documentation

H.56.2.1 ExtendedFeatureSet() [1/2]

```
BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeatureSet::ExtendedFeatureSet (
            const std::string & filename,
            int recordNumber)
```

Construct an AN2K11 EFS object from file data.

The file contains a complete ANSI/NIST record, and an object of this class represents a single Type-9 extended feature set structure.

Parameters

in	*filename*	The name of the file containing the complete ANSI/↩ NIST record.

Parameters

in	*recordNumber*	Which fingerprint minutiae record to read from the complete AN2K record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named file does not exist.
Error::StrategyError (p. 789)	An error occurred when opening or reading from the file.
Error::DataError (p. 390)	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for t

H.56.2.2 ExtendedFeatureSet() [2/2]

```
BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeatureSet::ExtendedFeatureSet (
                Memory::uint8Array & buf,
              int recordNumber)
```

Construct an AN2K11 EFS 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 Type-9 extended feature set structure.

Parameters

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

Parameters

in	*recordNumber*	Which fingerprint minutiae record to read from the complete AN2K record.

Exceptions

Error::DataError *(p. 390)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the reques

H.56.3 Member Function Documentation

H.56.3.1 getCPS()

`BiometricEvaluation::Feature::AN2K11EFS::CorePointSet BiometricEvaluation::Feature::AN2K11↩ EFS::ExtendedFeatureSet::getCPS () const`

Obtain the core point set.

The set may be empty as this Type-9 field is optional.

Returns

The set of core points.

H.56.3.2 getDPS()

`BiometricEvaluation::Feature::AN2K11EFS::DeltaPointSet BiometricEvaluation::Feature::AN2K11↩ EFS::ExtendedFeatureSet::getDPS () const`

Obtain the delta point set.

The set may be empty as this Type-9 field is optional.

Returns

The set of delta points.

H.56.3.3 getEAA()

ExaminerAnalysisAssessment `BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeatureSet::get↩ EAA () const`

Obtain the examiner's analysis assessment of the print.

Returns

Examiner's analysis assessment.

H.56.3.4 getImageInfo()

ImageInfo BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeatureSet::getImageInfo () const

Obtain the structure containing information about the image and Extended **Feature** (p. 115) Set.

Returns

The information about the image.

H.56.3.5 getLPM()

std::vector< LatentProcessingMethod > BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeature↩
Set::getLPM () const

Obtain set of methods used to process this latent.

The set may be empty as this Type-9 field is optional.

Returns

The set of latent processing methods.

H.56.3.6 getLSB()

Substrate BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeatureSet::getLSB () const

Returns

Substrate/surface on which the impression was deposited.

H.56.3.7 getMPS()

BiometricEvaluation::Feature::AN2K11EFS::MinutiaPointSet BiometricEvaluation::Feature::AN2↩
K11EFS::ExtendedFeatureSet::getMPS () const

Obtain the minutiae point set.

The set may be empty as this Type-9 field is optional.

Returns

The set of minutia points.

H.56.3.8 getMRCI()

BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCountInfo BiometricEvaluation::Feature↩
::AN2K11EFS::ExtendedFeatureSet::getMRCI () const

Obtain all the information relating to minutiae ridge count information.

Some of the information may not be present for the optional fields in the AN2k11 extended feature set.

Returns

The minutiae ridge count information structure.

H.56.3.9 getNFP()

BiometricEvaluation::Feature::AN2K11EFS::NoFeaturesPresent BiometricEvaluation::Feature::↩
AN2K11EFS::ExtendedFeatureSet::getNFP () const

Obtain the No Features Present indicators.

Returns

The flags for No Features Present.

H.56.3.10 getPAT()

std::vector< **Pattern** > BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeatureSet::getPAT ()
const

Returns

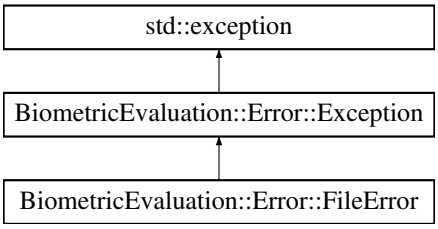
Collection of possible pattern classifications.

H.57 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.57.1 Detailed Description

File error when opening, reading, writing, etc.

H.57.2 Constructor & Destructor Documentation

H.57.2.1 FileError() [1/2]

```
BiometricEvaluation::Error::FileError::FileError ()
```

Construct a **FileError** (p. 420) object with the default information string.

H.57.2.2 FileError() [2/2]

```
BiometricEvaluation::Error::FileError::FileError (
            const std::string & info)
```

Construct a **FileError** (p. 420) object with an information string appended to the default information string.

H.58 BiometricEvaluation::IO::FileLogCabinet Class Reference

A class to represent a collection of log sheets.

`#include <be_io_filelogcabinet.h>`

Public Member Functions

- **FileLogCabinet** (const std::string &pathname, const std::string &description)
- **FileLogCabinet** (const std::string &pathname)
- std::shared_ptr< **FileLogsheet** > **newLogsheet** (const std::string &name, const std::string &description)
- std::string **getPathname** ()
- std::string **getDescription** ()
- unsigned int **getCount** ()

H.58.1 Detailed Description

A class to represent a collection of log sheets.

All **IO::FileLogsheet** (p. 424) backing files are stored in the directory managed by objects of this class.

H.58.2 Constructor & Destructor Documentation

H.58.2.1 FileLogCabinet() [1/2]

```
BiometricEvaluation::IO::FileLogCabinet::FileLogCabinet (
            const std::string & pathname,
            const std::string & description)
```

Create a new **FileLogCabinet** (p. 421) in the file system.

Parameters

in	*pathname*	The pathname where the **FileLogCabinet** (p. 421) is to be created.
in	*description*	The text used to describe the cabinet.

Exceptions

Error::ObjectExists *(p. 637)*	The cabinet was previously created.
Error::StrategyError (p. 789)	An error occurred when using the underlying file system.

H.58.2.2 FileLogCabinet() [2/2]

```
BiometricEvaluation::IO::FileLogCabinet::FileLogCabinet (
            const std::string & pathname)
```

Open an existing **FileLogCabinet** (p. 421).

Parameters

in	*pathname*	The pathname where the **FileLogCabinet** (p. 421) is located.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The cabinet does not exist in the file system.
Error::StrategyError (p. 789)	An error occurred when using the underlying file system.

H.58.3 Member Function Documentation

H.58.3.1 getCount()

```
unsigned int BiometricEvaluation::IO::FileLogCabinet::getCount ()
```

Obtain the number of items in the **FileLogCabinet** (p. 421).

@ returns The number of logsheets manages by the cabinet.

H.58.3.2 getDescription()

```
std::string BiometricEvaluation::IO::FileLogCabinet::getDescription ()
```

Obtain the description of the **FileLogCabinet** (p. 421).

@ returns The description of the **FileLogCabinet** (p. 421).

H.58.3.3 getPathname()

```
std::string BiometricEvaluation::IO::FileLogCabinet::getPathname ()
```

Obtain the pathname of the **FileLogCabinet** (p. 421).

@ returns The pathname of the **FileLogCabinet** (p. 421).

H.58.3.4 newLogsheet()

```
std::shared_ptr< FileLogsheet > BiometricEvaluation::IO::FileLogCabinet::newLogsheet (
            const std::string & name,
            const std::string & description)
```

Create a new **FileLogsheet** (p. 424) within the cabinet.

Parameters

in	*name*	The name of the **FileLogsheet** (p. 424) to be created. This can not be a path name.

Parameters

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

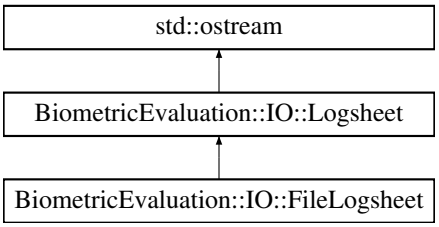
Error::ObjectExists (p. 637)	The sheet was previously created.
Error::StrategyError (p. 789)	An error occurred when using the underlying file system.

H.59 BiometricEvaluation::IO::FileLogsheet Class Reference

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

`#include <be_io_filelogsheet.h>`

Inheritance diagram for BiometricEvaluation::IO::FileLogsheet:



Public Member Functions

- **FileLogsheet** (const std::string &url, const std::string &description)

  Create a new log sheet.

- **FileLogsheet** (const std::string &url)

  Open an existing log sheet for appending.
- ~**FileLogsheet** ()
- std::string **sequence** (bool allEntries=false, bool **trim**=true, int32_t cursor= **BE_FILELOGSHEET_SEQ_NEXT**)

  *Sequence through a **FileLogsheet** (p. 424), returning one entry per invocation.*
- void **write** (const std::string &entry)

  Write a string as an entry to the backing store.
- void **writeComment** (const std::string &entry)

  Write a string as a comment to the backing store.
- void **writeDebug** (const std::string &entry)

  Write a string as a debug entry to the backing store.
- void **sync** ()

  Synchronize any buffered data to the underlying backing store.

Public Member Functions inherited from BiometricEvaluation::IO::Logsheet

- **Logsheet** ()

  *Create a **Logsheet** (p. 585) that has no backing store. A log entry is maintained, but cannot be permanently stored. This is the Null **Logsheet** (p. 585).*
- virtual ~**Logsheet** ()
- void **newEntry** ()

  Start a new entry, causing the existing entry to be closed and written.
- std::string **getCurrentEntry** () const

  Obtain the contents of the current entry currently under construction.
- void **resetCurrentEntry** ()
- uint32_t **getCurrentEntryNumber** () const

  Obtain the current entry number.
- void **setCommit** (const bool state)

  Enable or disable the commitment of normal entries to the backing log storage.
- bool **getCommit** () const

  Get the current entry commit state.
- void **setDebugCommit** (const bool state)

  Enable or disable the commitment of debug entries to the backing log storage.
- bool **getDebugCommit** () const

  Get the current debug entry commit state.
- void **setCommentCommit** (const bool state)

  Enable or disable the commitment of comment entries to the backing log storage.
- bool **getCommentCommit** () const

  Get the current comment entry commit state.
- void **setAutoSync** (bool state)
- bool **getAutoSync** () const

Static Public Member Functions

- static void **mergeLogsheets** (std::vector< std::shared_ptr< **FileLogsheet** > > &logsheets)

  *Merge multiple FileLogsheets into a single **FileLogsheet** (p. 424).*
- static std::string **trim** (const std::string &entry)

  *Trim delimiters from **FileLogsheet** (p. 424) entries.*

Static Public Member Functions inherited from BiometricEvaluation::IO::Logsheet

- static **Logsheet::Kind** **getTypeFromURL** (const std::string &url)

  *Map the URL scheme, taken from a string containing the entire URL, into a **Logsheet** (p. 585) type.*
- static bool **lineIsEntry** (const std::string &line)

  Helper function to determine whether a string is a valid log entry.
- static bool **lineIsComment** (const std::string &line)

  Helper function to determine whether a string is a valid comment log entry.
- static bool **lineIsDebug** (const std::string &line)

  Helper function to determine whether a string is a valid debug log entry.
- static std::string **trim** (const std::string &entry)

  *Trim delimiters from **Logsheet** (p. 585) entries.*

Static Public Attributes

- static const int32_t **BE_FILELOGSHEET_SEQ_START** = 1
- static const int32_t **BE_FILELOGSHEET_SEQ_NEXT** = 2

Static Public Attributes inherited from BiometricEvaluation::IO::Logsheet

- static const char **CommentDelimiter** = '#'
- static const char **EntryDelimiter** = 'E'
- static const char **DebugDelimiter** = 'D'
- static const std::string **DescriptionTag**
- static const std::string **FILEURLSCHEME**
- static const std::string **SYSLOGURLSCHEME**

Protected Member Functions

- **FileLogsheet** (const **FileLogsheet** &)
- **FileLogsheet** & **operator=** (const **FileLogsheet** &)
- void **updateCursor** ()

  Update the cursor position of the sequence file.

Protected Member Functions inherited from BiometricEvaluation::IO::Logsheet

- void **incrementEntryNumber** ()

  Increment the current entry number.
- std::string **getCurrentEntryNumberAsString** () const

  Obtain the current entry 'tag', in 'Edddd' format.

Protected Attributes

- std::unique_ptr< std::fstream > **_theLogFile**
- std::shared_ptr< std::fstream > **_sequenceFile**
- std::streamoff **_cursor**

Additional Inherited Members

Public Types inherited from BiometricEvaluation::IO::Logsheet

- enum class **Kind** { **Null** , **File** , **Syslog** }

H.59.1 Detailed Description

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

A **FileLogsheet** (p. 424) object can be constructed and passed back to the client by the LogCabinet object. All sheets created in this manner are placed in a common area maintained by the cabinet.

H.59.2 Constructor & Destructor Documentation

H.59.2.1 FileLogsheet() [1/3]

```
BiometricEvaluation::IO::FileLogsheet::FileLogsheet (
            const std::string & url,
            const std::string & description)
```

Create a new log sheet.

the log sheet is named by the uniform resource locator, usually starting with ' `file://`'. However, relative and absolute path names are also accepted for backward compatibility.

Parameters

in	*url*	The Uniform Resource Locator of the **File↩ Logsheet** (p. 424) to be created.

Parameters

in	*description*	The text used to describe the sheet. This text is written into the log file prior to any entries.

Exceptions

Error::ParameterError (p. *655*)	The URL is malformed.
Error::ObjectExists (p. *637*)	The sheet was previously created.
Error::StrategyError (p. *789*)	An error occurred when using the underlying file system, or name or parentDir is malformed.

H.59.2.2 FileLogsheet() [2/3]

```
BiometricEvaluation::IO::FileLogsheet::FileLogsheet (
            const std::string & url)
```

Open an existing log sheet for appending.

On open, the current entry counter is set to the last entry number plus one.

Note

Opening a large **FileLogsheet** (p. 424) may be a costly operation.

Parameters

in	*url*	The Uniform Resource Locator of the **FileLogsheet** (p. 424) to be opened.

Exceptions

Error::ParameterError *(p. 655)*	The URL is malformed.
Error::ObjectDoesNotExist *(p. 637)*	The sheet does not exist.
Error::StrategyError *(p. 789)*	An error occurred when using the underlying file system, or name or parentDir is malform

H.59.2.3 ~FileLogsheet()

```
BiometricEvaluation::IO::FileLogsheet::~FileLogsheet ()
```

Destructor

H.59.2.4 FileLogsheet() [3/3]

```
BiometricEvaluation::IO::FileLogsheet::FileLogsheet (
            const  FileLogsheet & )  [protected]
```

Prevent copying of **FileLogsheet** (p. 424) objects

H.59.3 Member Function Documentation

H.59.3.1 mergeLogsheets()

```
static void BiometricEvaluation::IO::FileLogsheet::mergeLogsheets (
            std::vector< std::shared_ptr<  FileLogsheet > > & logsheets)  [static]
```

Merge multiple FileLogsheets into a single **FileLogsheet** (p. 424).
Logsheet (p. 585) 2 - n will be appended to **Logsheet** (p. 585) 1.

Parameters

logSheets	**Logsheet** (p. 585) to merge.

Exceptions

Error::FileError *(p. 420)*	**Error** (p. 112) during log sequence.
Error::StrategyError *(p. 789)*	**Error** (p. 112) during log sequence.

H.59.3.2 operator=()

```
FileLogsheet & BiometricEvaluation::IO::FileLogsheet::operator= (
            const FileLogsheet & ) [protected]
```

Prevent copying of **FileLogsheet** (p. 424) objects

H.59.3.3 sequence()

```
std::string BiometricEvaluation::IO::FileLogsheet::sequence (
            bool allEntries = false,
            bool trim = true,
            int32_t cursor = BE_FILELOGSHEET_SEQ_NEXT)
```

Sequence through a **FileLogsheet** (p. 424), returning one entry per invocation.

Parameters

allEntries	Include debgug and comment entries when sequencing
trim	Whether or not to include entry delimiters.
cursor	The location within the sequence to return.

Returns

The contents of the sequenced entry, as was originally given to **write()** (p. 432).

Exceptions

Error::FileError (p. 420),***Error*** (p. 112)	occured while performing file **IO** (p. 136).
Error::ObjectDoesNotExist (p. 637)	The **FileLogsheet** (p. 424) cannot be found on disk.
Error::StrategyError (p. 789)	Invalid cursor position or the contents of the **FileLogsheet** (p. 424) is malformed.

H.59.3.4 sync()

```
void BiometricEvaluation::IO::FileLogsheet::sync ()  [virtual]
```

Synchronize any buffered data to the underlying backing store.
This syncing is dependent on the behavior of the underlying storage mechanism.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 593).

H.59.3.5 trim()

```
static std::string BiometricEvaluation::IO::FileLogsheet::trim (
            const std::string & entry)  [static]
```

Trim delimiters from **FileLogsheet** (p. 424) entries.
Works for comments and numbered entries.

Parameters

in	*entry*	The entry to trim.

Returns

Delimiter-less entry.

H.59.3.6 updateCursor()

```
void BiometricEvaluation::IO::FileLogsheet::updateCursor ()  [protected]
```

Update the cursor position of the sequence file.

Exceptions

Error::FileError (p. 420)	**Error** (p. 112) getting file position from sequence file.

H.59.3.7 write()

```
void BiometricEvaluation::IO::FileLogsheet::write (
            const std::string & entry)  [virtual]
```

Write a string as an entry to the backing store.

This does not affect the current log entry buffer, but does increment the entry number.

Parameters

in	*entry*	The text of the log entry.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 594).

H.59.3.8 writeComment()

```
void BiometricEvaluation::IO::FileLogsheet::writeComment (
            const std::string & entry)  [virtual]
```

Write a string as a comment to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

in	*entry*	The text of the comment.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 594).

H.59.3.9 writeDebug()

```
void BiometricEvaluation::IO::FileLogsheet::writeDebug (
            const std::string & entry)  [virtual]
```

Write a string as a debug entry to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A debug line is prefixed with DebugDelimiter followed by a space.

Parameters

in	*entry*	The text of the debug message.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when logging.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 595).

H.59.4 Member Data Documentation

H.59.4.1 _cursor

`std::streamoff BiometricEvaluation::IO::FileLogsheet::_cursor [protected]`

Position of the sequencer, relative to SOF

H.59.4.2 _sequenceFile

`std::shared_ptr<std::fstream> BiometricEvaluation::IO::FileLogsheet::_sequenceFile [protected]`

Stream used for sequencing

H.59.4.3 _theLogFile

`std::unique_ptr<std::fstream> BiometricEvaluation::IO::FileLogsheet::_theLogFile [protected]`

Stream used for writing the log file

H.59.4.4 BE_FILELOGSHEET_SEQ_NEXT

`const int32_t BiometricEvaluation::IO::FileLogsheet::BE_FILELOGSHEET_SEQ_NEXT = 2 [static]`

Sequence from current position

H.59.4.5 BE_FILELOGSHEET_SEQ_START

`const int32_t BiometricEvaluation::IO::FileLogsheet::BE_FILELOGSHEET_SEQ_START = 1 [static]`

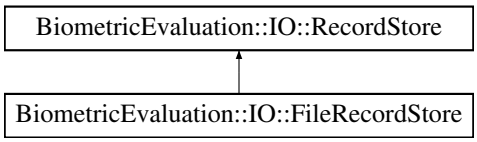
Sequence from beginning

H.60 BiometricEvaluation::IO::FileRecordStore Class Reference

A class to represent the record store data storage mechanism implemented as files for each record.

`#include <be_io_filerecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::FileRecordStore:



Public Member Functions

- **FileRecordStore** (const std::string &pathname, const std::string &description)
- **FileRecordStore** (const std::string &name, **IO::Mode** mode= **IO::Mode::ReadOnly**)
- void **insert** (const std::string &key, const void *const data, const uint64_t size) override
- void **remove** (const std::string &key) override
- **Memory::uint8Array** **read** (const std::string &key) const override

  Read a complete record from a store.
- void **replace** (const std::string &key, const void *const data, const uint64_t size) override final
- uint64_t **length** (const std::string &key) const override
- void **flush** (const std::string &key) const override
- **RecordStore::Record** **sequence** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  *Move the **RecordStore** (p. 700).*
- uint64_t **getSpaceUsed** () const override

  Obtain real storage utilization.
- void **sync** () const override
- unsigned int **getCount** () const override
- std::string **getPathname** () const override
- std::string **getDescription** () const override
- void **changeDescription** (const std::string &description) override
- **FileRecordStore** (const **FileRecordStore** &)=delete
- **FileRecordStore** & **operator=** (const **FileRecordStore** &)=delete
- virtual void **insert** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const **Memory::uint8Array** &data)

Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- virtual bool **containsKey** (const std::string &key) const

  *Determines whether the **RecordStore** (p. 700) contains an element with the specified key.*
- virtual **iterator** **begin** () noexcept
- virtual **iterator** **end** () noexcept

Additional Inherited Members

Public Types inherited from BiometricEvaluation::IO::RecordStore

- enum class **Kind** {
  BerkeleyDB , **Archive** , **File** , **SQLite** ,
  Compressed , **List** , **Default** = BerkeleyDB }
- using **Record** = struct Record
- using **iterator** = **IO::RecordStoreIterator**

Static Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- static bool **isRecordStore** (const std::string &pathname)

  *Determine if a location appears to be a **RecordStore** (p. 700).*
- static std::shared_ptr< **RecordStore** > **openRecordStore** (const std::string &pathname, **IO::Mode** mode= **Mode::ReadOnly**)

  *Open an existing **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static std::shared_ptr< **RecordStore** > **createRecordStore** (const std::string &pathname, const std::string &description, const **IO::RecordStore::Kind** &kind)

  *Create a new **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static void **removeRecordStore** (const std::string &pathname)
- static void **mergeRecordStores** (const std::string &mergePathname, const std::string &description, const **IO::RecordStore::Kind** &kind, const std::vector< std::string > &pathnames, const std::function< bool()> &interrupt=[ ]() {return(false);})

  *Create a new **RecordStore** (p. 700) that contains the contents of several other RecordStores.*

Static Public Attributes inherited from BiometricEvaluation::IO::RecordStore

- static const std::string **INVALIDKEYCHARS**
- static const int **BE_RECSTORE_SEQ_START** = 1
- static const int **BE_RECSTORE_SEQ_NEXT** = 2

H.60.1 Detailed Description

A 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** (p. 789) will be thrown if the key string is not compliant. A **FileRecordStore** (p. 433) has the additional requirement that a key name may not contain path delimiter characters ('/' and '\'), or begin with whitespace.

H.60.2 Constructor & Destructor Documentation

H.60.2.1 FileRecordStore() [1/2]

```
BiometricEvaluation::IO::FileRecordStore::FileRecordStore (
            const std::string & pathname,
            const std::string & description)
```

Create a new **FileRecordStore** (p. 433), read/write mode.

Parameters

in	*pathname*	The directory where the store is to be created.
in	*description*	The store's description.

Exceptions

Error::ObjectExists (p. 637)	The store already exists.
Error::StrategyError (p. 789)	An error occurred when accessing the underlying file system.

H.60.2.2 FileRecordStore() [2/2]

```
BiometricEvaluation::IO::FileRecordStore::FileRecordStore (
            const std::string & name,
             IO::Mode mode = IO::Mode::ReadOnly)
```

Open an existing **FileRecordStore** (p. 433).

Parameters

in	*name*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The store does not exist.
Error::StrategyError (p. 789)	An error occurred when accessing the underlying file system.

H.60.3 Member Function Documentation

H.60.3.1 changeDescription()

```
void BiometricEvaluation::IO::FileRecordStore::changeDescription (
            const std::string & description)  [override], [virtual]
```

Change the description of the **RecordStore** (p. 700).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 703).

H.60.3.2 flush()

```
void BiometricEvaluation::IO::FileRecordStore::flush (
            const std::string & key) const  [override], [virtual]
```

Commit the record's data to storage.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 704).

H.60.3.3 getCount()

```
unsigned int BiometricEvaluation::IO::FileRecordStore::getCount () const  [override], [virtual]
```

Obtain the number of items in the **RecordStore** (p. 700).

Returns

The number of items in the **RecordStore** (p. 700).

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.60.3.4 getDescription()

```
std::string BiometricEvaluation::IO::FileRecordStore::getDescription () const  [override], [virtual]
```

Obtain a textual description of the **RecordStore** (p. 700).

Returns

The **RecordStore** (p. 700)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.60.3.5 getPathname()

```
std::string BiometricEvaluation::IO::FileRecordStore::getPathname () const  [override], [virtual]
```

Return the path name of the **RecordStore** (p. 700).

Returns

Where in the file system the **RecordStore** (p. 700) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.60.3.6 getSpaceUsed()

```
uint64_t BiometricEvaluation::IO::FileRecordStore::getSpaceUsed () const  [override], [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** (p. 700).

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 706).

H.60.3.7 insert() [1/2]

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

Insert a record into the store.

Parameters

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

Parameters

in	*data*	The data for the record.

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 706).

H.60.3.8 insert() [2/2]

```
void BiometricEvaluation::IO::FileRecordStore::insert (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [override], [virtual]
```

Insert a record into the store.

Parameters

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

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Implements **BiometricEvaluation::IO::RecordStore** (p. 707).

H.60.3.9 length()

```
uint64_t BiometricEvaluation::IO::FileRecordStore::length (
            const std::string & key) const  [override], [virtual]
```

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 708).

H.60.3.10 move()

```
void BiometricEvaluation::IO::FileRecordStore::move (
            const std::string & pathname)  [override], [virtual]
```

Move the **RecordStore** (p. 700).

The **RecordStore** (p. 700) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 700).

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 710).

H.60.3.11 read()

```
Memory::uint8Array BiometricEvaluation::IO::FileRecordStore::read (
            const std::string & key) const  [override], [virtual]
```

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

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

Returns

The record associated with the key.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 712).

H.60.3.12 remove()

```
void BiometricEvaluation::IO::FileRecordStore::remove (
            const std::string & key)  [override], [virtual]
```

Remove a record from the store.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 713).

H.60.3.13 replace() [1/2]

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

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.60.3.14 replace() [2/2]

```
void BiometricEvaluation::IO::FileRecordStore::replace (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [final], [override], [virtual]
```

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Parameters

in	*size*	The size of the record, in bytes.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.60.3.15 sequence()

```
RecordStore::Record BiometricEvaluation::IO::FileRecordStore::sequence (
        int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the function to return the next record. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The record that is currently in sequence.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 715).

H.60.3.16 sequenceKey()

```
std::string BiometricEvaluation::IO::FileRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key.

Sequencing means to start at some point in the store and return the key, then repeatedly calling the function to return the next key. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The key of the currently sequenced record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 716).

H.60.3.17 setCursorAtKey()

```
void BiometricEvaluation::IO::FileRecordStore::setCursorAtKey (
            const std::string & key) [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 700), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 443).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 443).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.60.3.18 sync()

```
void BiometricEvaluation::IO::FileRecordStore::sync () const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.61 BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem Struct Reference

Representation of information about a fingerprint reader system.

```
#include <be_feature_an2k7minutiae.h>
```

Public Attributes

- std::string **name**
- **EncodingMethod method**
- std::string **equipment**

H.61.1 Detailed Description

Representation of information about a fingerprint reader system.

H.61.2 Member Data Documentation

H.61.2.1 equipment

`std::string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::equipment`

Optional ID for equipment used in system

H.61.2.2 method

`EncodingMethod` `BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::method`

Method used to encoded minutiae

H.61.2.3 name

`std::string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::name`

Name for system that encoded minutiae

H.62 BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition Struct Reference

Locations of an individual finger segment in a slap.

`#include <be_finger_an2kview_capture.h>`

Public Member Functions

- **FingerSegmentPosition** (const **Finger::Position** **fingerPosition**, const Image::CoordinateSet **coordinates**)

  *Create an **FingerSegmentPosition** (p. 446) struct.*

Public Attributes

- **Finger::Position** **fingerPosition**
- Image::CoordinateSet **coordinates**

H.62.1 Detailed Description

Locations of an individual finger segment in a slap.

H.62.2 Constructor & Destructor Documentation

H.62.2.1 FingerSegmentPosition()

```
BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::FingerSegmentPosition (
            const  Finger::Position fingerPosition,
            const Image::CoordinateSet coordinates)
```

Create an **FingerSegmentPosition** (p. 446) struct.

Parameters

fingerPosition	**Finger** (p. 122) depicted in this segment.
coordinates	Collection of coordinates that compose the segment bonding polygon.

H.62.3 Member Data Documentation

H.62.3.1 coordinates

`Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::coordinates`

Points composing the segmented polygon

H.62.3.2 fingerPosition

`Finger::Position` `BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::fingerPosition`

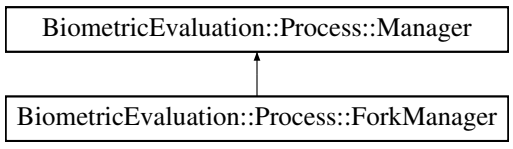
Finger (p. 122) depicted in this segment

H.63 BiometricEvaluation::Process::ForkManager Class Reference

Manager (p. 596) implementation that starts Workers by calling fork(2).

`#include <be_process_forkmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::ForkManager:



Public Member Functions

- **ForkManager** ()
- std::shared_ptr< **WorkerController** > **addWorker** (std::shared_ptr< **Worker** > worker)

  *Adds a **Worker** (p. 828) to be managed by this **Manager** (p. 596).*
- void **startWorkers** (bool wait=true, bool communicate=false)

  *Begin **Worker** (p. 828)'s work.*
- void **startWorker** (std::shared_ptr< **WorkerController** > worker, bool wait=true, bool communicate=false)

  Start a worker.
- void **stopWorker** (std::shared_ptr< **WorkerController** > workerController)

  *Ask **Worker** (p. 828) 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** (p. 447) is responsbile for a particular PID.*
- void **setNotWorking** (const pid_t pid)

  Set Status.isWorking for PID to false.
- void **markAllFinished** ()

  *Call **setNotWorking()** (p. 453) for all PIDs known to this **ForkManager** (p. 447).*
- bool **getIsWorkingStatus** (const pid_t pid) const

  Get Status.isWorking for PID.
- void **waitForWorkerExit** ()

  Block until all Workers have exited.
- ~**ForkManager** ()

  ***ForkManager** (p. 447) destructor.*
- void **setExitCallback** (void(*exitCallback)(std::shared_ptr< **ForkWorkerController** > worker, int stat_loc))

  Call a function in your program when a child exits.
- void **setExitStatus** (const pid_t pid, const int32_t waitStatus)

  *Set the exit status in the **WorkerController** (p. 834) for given process ID.*

Public Member Functions inherited from BiometricEvaluation::Process::Manager

- **Manager** ()

  ***Manager** (p. 596) constructor.*
- virtual uint32_t **getNumCompletedWorkers** () const

  Obtain the number of Workers that have exited.
- virtual uint32_t **getNumActiveWorkers** () const

  Obtain the number of Workers that are still working.
- virtual uint32_t **getTotalWorkers** () const

  Obtain the number of Workers this class is handling.
- virtual void **reset** ()

  Reuse all Workers.
- virtual bool **waitForMessage** (std::shared_ptr< **WorkerController** > &sender, int *nextFD=nullptr, int numSeconds=-1) const

*Wait for a message from a **Worker*** (p. 828).

- virtual bool **getNextMessage** (std::shared_ptr< **WorkerController** > &sender, **Memory::uint8Array** &message, int numSeconds=-1) const

  *Obtain a message from a **Worker*** (p. 828).
- virtual void **broadcastMessage** (**Memory::uint8Array** &message) const

  Send one message to all Workers.
- virtual ~**Manager** ()

  Manager (p. 596) *destructor.*

Static Public Member Functions

- static void **defaultExitCallback** (std::shared_ptr< **ForkWorkerController** > worker, int status)

  A default exit callback function.

Static Public Attributes

- static std::list< **ForkManager** * > **FORKMANAGERS**

  List of all instantiated ForkManagers.

Additional Inherited Members

Protected Attributes inherited from BiometricEvaluation::Process::Manager

- std::vector< std::shared_ptr< **WorkerController** > > **_workers**
- std::vector< std::shared_ptr< **WorkerController** > > **_pendingExit**

H.63.1 Detailed Description

Manager (p. 596) implementation that starts Workers by calling fork(2).

H.63.2 Constructor & Destructor Documentation

H.63.2.1 ForkManager()

```
BiometricEvaluation::Process::ForkManager::ForkManager ()
```

ForkManager (p. 447) constructor.

H.63.3 Member Function Documentation

H.63.3.1 addWorker()

```
std::shared_ptr< WorkerController > BiometricEvaluation::Process::ForkManager::addWorker (
            std::shared_ptr< Worker > worker) [virtual]
```

Adds a **Worker** (p. 828) to be managed by this **Manager** (p. 596).

Parameters

worker	A **Worker** (p. 828) instance to run.

Returns

shared_ptr to worker.

Implements **BiometricEvaluation::Process::Manager** (p. 597).

H.63.3.2 broadcastSignal()

```
void BiometricEvaluation::Process::ForkManager::broadcastSignal (
            int signo)
```

Send a POSIX signal to all workers.

Parameters

in	*signo*	The signal to send.

H.63.3.3 defaultExitCallback()

```
static void BiometricEvaluation::Process::ForkManager::defaultExitCallback (
            std::shared_ptr< ForkWorkerController > worker,
            int status) [static]
```

A default exit callback function.
Writes to stdout in the form: PID #: Exited .

Parameters

worker	The **ForkWorkerController** (p. 456) object that exited.
status	The status of the **Worker** (p. 828) that exited (from wait(2)).

H.63.3.4 getIsWorkingStatus()

```
bool BiometricEvaluation::Process::ForkManager::getIsWorkingStatus (
            const pid_t pid) const
```

Get Status.isWorking for PID.

Parameters

in	*pid*	PID whose inWorking flag should be queried

Exceptions

Error::ObjectDoesNotExist (p. 637)	PID not under this manager's control.

H.63.3.5 responsibleFor()

```
bool BiometricEvaluation::Process::ForkManager::responsibleFor (
            const pid_t pid) const
```

Obtain whether or not this **ForkManager** (p. 447) is responsbile for a particular PID.

Parameters

in	*pid*	PID in question

Returns

true if this **ForkManager** (p. 447) spawned pid, false otherwise.

H.63.3.6 setExitCallback()

```
void BiometricEvaluation::Process::ForkManager::setExitCallback (
            void(* exitCallback )(std::shared_ptr< ForkWorkerController > worker, int stat_loc))
```

Call a function in your program when a child exits.

Parameters

exitCallback	Function pointer to a method that takes a shared_ptr to a **ForkWorkerController** (p. 456) and the integer status information.

Note

The exit callback will not have any effect if the **Manager** (p. 596) is not set to wait for Workers.

H.63.3.7 setExitStatus()

```
void BiometricEvaluation::Process::ForkManager::setExitStatus (
            const pid_t pid,
            const int32_t waitStatus)
```

Set the exit status in the **WorkerController** (p. 834) for given process ID.

Parameters

in	*pid*	PID whose exit status should be set.
in	*status*	Status, as returned from wait(2).

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	PID not under this manager's control.

Note

Exit status is only set if process exited cleanly.

H.63.3.8 setNotWorking()

```
void BiometricEvaluation::Process::ForkManager::setNotWorking (
            const pid_t pid)
```

Set Status.isWorking for PID to false.

Parameters

in	*pid*	PID whose isWorking flag should be set to false

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	PID not under this manager's control.

H.63.3.9 startWorker()

```
void BiometricEvaluation::Process::ForkManager::startWorker (
            std::shared_ptr< WorkerController > worker,
            bool wait = true,
            bool communicate = false)  [virtual]
```

Start a worker.

Parameters

	worker	Pointer to a **WorkerController** (p. 834) that is being managed by this **Manager** (p. 596) instance.
	wait	Whether or not to wait for this **Worker** (p. 828) to exit before returning control to the caller.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists (p. 637)	worker is already working.
Error::StrategyError (p. 789)	worker is not managed by this **Manager** (p. 596) instance.

Implements **BiometricEvaluation::Process::Manager** (p. 600).

H.63.3.10 startWorkers()

```
void BiometricEvaluation::Process::ForkManager::startWorkers (
            bool wait = true,
            bool communicate = false)  [virtual]
```

Begin **Worker** (p. 828)'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 *(p. 637)*	At least one **Worker** (p. 828) is already working.
Error::StrategyError *(p. 789)*	Problem forking.

Implements **BiometricEvaluation::Process::Manager** (p. 601).

H.63.3.11 stopWorker()

```
void BiometricEvaluation::Process::ForkManager::stopWorker (
            std::shared_ptr< WorkerController > workerController)  [virtual]
```

Ask **Worker** (p. 828) to exit.

Sends SIGUSR1 to the **Worker** (p. 828), which **ForkManager** (p. 447) will handle automatically.

Parameters

workerController	Pointer to the **ForkWorkerController** (p. 456) that should be stopped.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	worker is not working.
Error::StrategyError *(p. 789)*	Problem sending the signal.

Attention

Do not call **stopWorker()** (p. 455) 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** (p. 828) exits.

Implements **BiometricEvaluation::Process::Manager** (p. 602).

H.63.3.12 waitForWorkerExit()

```
void BiometricEvaluation::Process::ForkManager::waitForWorkerExit () [virtual]
```

Block until all Workers have exited.

Use this method if wait=false was set during a call to startWorker(s) but now wait=true is desired.

Implements **BiometricEvaluation::Process::Manager** (p. 604).

H.63.4 Member Data Documentation

H.63.4.1 FORKMANAGERS

```
std::list< ForkManager*> BiometricEvaluation::Process::ForkManager::FORKMANAGERS [static]
```

List of all instantiated ForkManagers.

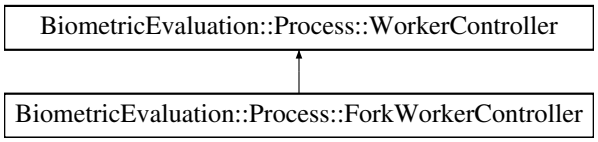
This is not a list of managed pointers to ForkManagers. If it was, the smart pointer's destructor would attempt to delete the object being pointed to at program termination, which is ultimately sometime after the destructor of the **ForkManager** (p. 447) itself was called.

H.64 BiometricEvaluation::Process::ForkWorkerController Class Reference

Wrapper of a **Worker** (p. 828) returned from a **Process::ForkManager** (p. 447).

```
#include <be_process_forkmanager.h>
```

Inheritance diagram for BiometricEvaluation::Process::ForkWorkerController:



Public Member Functions

- bool **isWorking** () const

  *Obtain whether or not **Worker** (p. 828) is working.*
- bool **everWorked** () const

  *Obtain whether or not this **Worker** (p. 828) has ever worked.*
- void **reset** ()

  *Reuse the **Worker** (p. 828).*
- pid_t **getPID** () const

  Obtain the PID of this process this instance represents.
- ~**ForkWorkerController** ()

  ***ForkWorkerController** (p. 456) destructor.*

Public Member Functions inherited from BiometricEvaluation::Process::WorkerController

- **WorkerController** (std::shared_ptr< **Worker** > worker)
- virtual void **sendMessageToWorker** (const **Memory::uint8Array** &message)

  *Send a message to the **Worker** (p. 828) contained within this **WorkerController** (p. 834).*
- virtual void **setParameter** (const std::string &name, std::shared_ptr< void > argument)

  *Set the parameter to be passed to the **Worker** (p. 828).*
- virtual void **setParameterFromDouble** (const std::string &name, double argument)

  *Set a double parameter to be passed to the **Worker** (p. 828).*
- virtual void **setParameterFromInteger** (const std::string &name, int64_t argument)

  *Set an integer parameter to be passed to the **Worker** (p. 828).*
- virtual void **setParameterFromString** (const std::string &name, const std::string &argument)

  *Set a string parameter to be passed to the **Worker** (p. 828).*
- bool **finishedWorking** () const

  *Obtain whether or not this **Worker** (p. 828) has both started and finished its task.*
- std::shared_ptr< **Worker** > **getWorker** () const

  *Obtain the **Worker** (p. 828) instance being wrapped.*
- virtual int32_t **getExitStatus** () const final

  *Obtain the exit status of the wrapped **Worker** (p. 828).*
- virtual ~**WorkerController** ()

  ***WorkerController** (p. 834) destructor.*

Static Public Member Functions

- static void **_stop** (int signal)

  Tell _staticWorker to stop.

Friends

- void **ForkManager::startWorkers** (bool wait, bool communicate)

  *Begin **Worker** (p. 828)'s work.*
- void **ForkManager::startWorker** (std::shared_ptr< **WorkerController** > worker, bool wait, bool communicate)

  *Restart a completed **Worker** (p. 828).*
- void **ForkManager::stopWorker** (std::shared_ptr< **WorkerController** > workerController)

  *Ask **Worker** (p. 828) to exit.*
- std::shared_ptr< **WorkerController** > **ForkManager::addWorker** (std::shared_ptr< **Worker** > worker)

  *Adds a **Worker** (p. 828) to be managed by this **Manager** (p. 596).*
- void **ForkManager::setExitStatus** (const pid_t pid, const int32_t waitStatus)

  *Set the exit status in the **WorkerController** (p. 834) for given process ID.*

Additional Inherited Members

Protected Attributes inherited from BiometricEvaluation::Process::WorkerController

- std::shared_ptr< **Worker** > **_worker**
- bool **_rvSet**
- int32_t **_rv**

H.64.1 Detailed Description

Wrapper of a **Worker** (p. 828) returned from a **Process::ForkManager** (p. 447).

H.64.2 Member Function Documentation

H.64.2.1 _stop()

```
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).

H.64.2.2 everWorked()

```
bool BiometricEvaluation::Process::ForkWorkerController::everWorked () const  [virtual]
```

Obtain whether or not this **Worker** (p. 828) has ever worked.

Returns

true the **Worker** (p. 828) has ever or is currently working, false otherwise.

Note

reset() (p. 459) will change the result of this method.

Implements **BiometricEvaluation::Process::WorkerController** (p. 836).

H.64.2.3 getPID()

```
pid_t BiometricEvaluation::Process::ForkWorkerController::getPID () const
```

Obtain the PID of this process this instance represents.

Returns

pid of the process this instance represents.

Note

Call isRunning() before doing anything with the PID returned from this function.

H.64.2.4 isWorking()

```
bool BiometricEvaluation::Process::ForkWorkerController::isWorking () const  [virtual]
```

Obtain whether or not **Worker** (p. 828) is working.

Returns

Whether or not the **Worker** (p. 828) is working.

Implements **BiometricEvaluation::Process::WorkerController** (p. 837).

H.64.2.5 reset()

```
void BiometricEvaluation::Process::ForkWorkerController::reset ()  [virtual]
```

Reuse the **Worker** (p. 828).

Exceptions

Error::ObjectExists (p. *637*)	The previously started **Worker** (p. 828) is still running.

Reimplemented from **BiometricEvaluation::Process::WorkerController** (p. 837).

H.64.3 Friends And Related Symbol Documentation

H.64.3.1 ForkManager::addWorker

```
std::shared_ptr< WorkerController > ForkManager::addWorker (
            std::shared_ptr< Worker > worker)  [friend]
```

Adds a **Worker** (p. 828) to be managed by this **Manager** (p. 596).

Parameters

worker	A **Worker** (p. 828) instance to run.

Returns

shared_ptr to worker.

H.64.3.2 ForkManager::setExitStatus

```
void ForkManager::setExitStatus (
            const pid_t pid,
            const int32_t waitStatus) [friend]
```

Set the exit status in the **WorkerController** (p. 834) for given process ID.

Parameters

in	*pid*	PID whose exit status should be set.
in	*status*	Status, as returned from wait(2).

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	PID not under this manager's control.

Note

Exit status is only set if process exited cleanly.

H.64.3.3 ForkManager::startWorker

```
void ForkManager::startWorker (
            std::shared_ptr< WorkerController > worker,
            bool wait,
            bool communicate) [friend]
```

Restart a completed **Worker** (p. 828).

Parameters

	worker	Pointer to a **WorkerController** (p. 834) that is being managed by this **Manager** (p. 596) instance.
	wait	Whether or not to wait for this **Worker** (p. 828) to exit before returning control to the caller.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists (p. 637)	worker is already working.
Error::StrategyError (p. 789)	worker is not managed by this **Manager** (p. 596) instance.

H.64.3.4 ForkManager::startWorkers

```
void ForkManager::startWorkers (
            bool wait,
            bool communicate) [friend]
```

Begin **Worker** (p. 828)'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 (p. 637)	One or more of the Workers is already working.
Error::StrategyError (p. 789)	Problem forking.

H.64.3.5 ForkManager::stopWorker

```
void ForkManager::stopWorker (
            std::shared_ptr< WorkerController > workerController) [friend]
```

Ask **Worker** (p. 828) to exit.

Sends SIGUSR1 to the **Worker** (p. 828), which **ForkManager** (p. 447) will handle automatically.

Parameters

workerController	Pointer to the **ForkWorkerController** (p. 456) that should be stopped.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	worker is not working.
Error::StrategyError *(p. 789)*	Problem sending the signal.

H.65 BiometricEvaluation::Feature::AN2K11EFS::FPPPosition Struct Reference

Representation of finger-palm-plantar position.

`#include <be_feature_an2k11efs.h>`

Public Attributes

- **Feature::FGP fgp**
- bool **has_fsm**
- FingerprintSegment **fsm**
- bool **has_ocf**
- OffCenterFingerPosition **ocf**
- bool **has_sgp**
- BiometricEvaluation::Image::CoordinateSet **sgp**

H.65.1 Detailed Description

Representation of finger-palm-plantar position.

Contains one or more possible physical positions that correspond to the region of interest. Clients of this structure must check the fgp value to determine which of the position codes (Finger/Palm/Plantar) applies.

H.65.2 Member Data Documentation

H.65.2.1 fgp

Feature::FGP BiometricEvaluation::Feature::AN2K11EFS::FPPPosition::fgp

The friction ridge generalized position

H.65.2.2 fsm

`FingerprintSegment BiometricEvaluation::Feature::AN2K11EFS::FPPPosition::fsm`

The finger segment position

H.65.2.3 ocf

`OffCenterFingerPosition BiometricEvaluation::Feature::AN2K11EFS::FPPPosition::ocf`

The off-center fingerprint position

H.65.2.4 sgp

`BiometricEvaluation::Image::CoordinateSet BiometricEvaluation::Feature::AN2K11EFS::FPPPosition::sgp`

The segment polygon

H.66 BiometricEvaluation::Video::Frame Struct Reference

Public Attributes

- **Image::Size size**
- int64_t **timestamp**
- **Memory::uint8Array data**

H.67 BiometricEvaluation::Feature::FrictionRidgeGeneralized Position Struct Reference

Representation of the position (Finger/Palm/Plantar) used in this class and child classes.

`#include <be_feature.h>`

Public Attributes

- **PositionType posType**
- union {
  Finger::Position fingerPos
  Palm::Position palmPos
  Plantar::Position plantarPos
  } **position**

H.67.1 Detailed Description

Representation of the position (Finger/Palm/Plantar) used in this class and child classes.

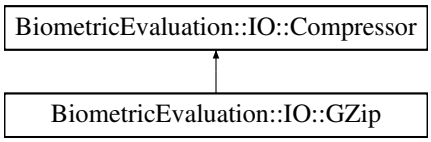
When the AN2K11 FGP field is read, it may represent a finger, palm, or plantar position. The union is tagged to indicate which position is present.

H.68 BiometricEvaluation::IO::GZip Class Reference

An **IO::Compressor** (p. 360) for gzip compression from zlib.

`#include <be_io_gzip.h>`

Inheritance diagram for BiometricEvaluation::IO::GZip:



Public Member Functions

- **Memory::uint8Array compress** (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize) const

  Compress a buffer.
- **Memory::uint8Array compress** (const **Memory::uint8Array** &uncompressedData) const

  Compress a buffer.
- void **compress** (const uint8_t *const uncompressedData, uint64_t uncompressedDataSize, const std::string &outputFile) const

  Compress a buffer.
- void **compress** (const **Memory::uint8Array** &uncompressedData, const std::string &outputFile) const

  Compress a buffer.
- **Memory::uint8Array compress** (const std::string &inputFile) const

  Compress a file.
- void **compress** (const std::string &inputFile, const std::string &outputFile) const

  Compress a file.
- **Memory::uint8Array decompress** (const uint8_t *const compressedData, uint64_t compressedDataSize) const

  Decompress a compressed buffer.
- **Memory::uint8Array decompress** (const **Memory::uint8Array** &compressedData) const

  Decompress a compressed buffer.
- **Memory::uint8Array decompress** (const std::string &input) const

  Decompress a compressed buffer into a file.
- void **decompress** (const std::string &inputFile, const std::string &outputFile) const

  Decompress a file.
- void **decompress** (const uint8_t *const compressedData, const uint64_t compressedDataSize, const std::string &outputFile) const

  Decompress a file.
- void **decompress** (const **Memory::uint8Array** &compressedData, const std::string &outputFile) const

  Decompress a file.
- **GZip** (const **GZip** &other)=delete

  Copy constructor (disabled).
- **GZip** & **operator=** (const **GZip** &other)=delete

  Assignment overload (disabled).

Public Member Functions inherited from BiometricEvaluation::IO::Compressor

- **Compressor** ()

  *Create a new **Compressor** (p. 360) object.*
- void **setOption** (const std::string &optionName, const std::string &optionValue)

  Assign a compressor option.
- void **setOption** (const std::string &optionName, int64_t optionValue)

  Assign a compressor option.
- std::string **getOption** (const std::string &optionName) const

  Obtain a compressor option as an integer.
- int64_t **getOptionAsInteger** (const std::string &optionName) const

  Obtain a compressor option as an integer.
- void **removeOption** (const std::string &optionName)

  Remove a compressor option.
- virtual ~**Compressor** ()
- **Compressor** (const **Compressor** &other)=delete

  Copy constructor (disabled).
- **Compressor** & **operator=** (const **Compressor** &other)=delete

  Assignment overload (disabled).

Static Public Attributes

- static const std::string **COMPRESSION_LEVEL**
- static const std::string **COMPRESSION_STRATEGY**
- static const std::string **COMPRESSION_METHOD**
- static const std::string **INPUT_DATA_TYPE**
- static const std::string **WINDOW_BITS**
- static const std::string **MEMORY_LEVEL**
- static const std::string **CHUNK_SIZE**

Additional Inherited Members

Public Types inherited from BiometricEvaluation::IO::Compressor

- enum class **Kind** { **GZIP** }

Static Public Member Functions inherited from BiometricEvaluation::IO::Compressor

- static std::shared_ptr< **Compressor** > **createCompressor** (**Compressor::Kind** compressorKind=Kind↩ ::GZIP)

H.68.1 Detailed Description

An **IO::Compressor** (p. 360) for gzip compression from zlib.

H.68.2 Constructor & Destructor Documentation

H.68.2.1 GZip()

```
BiometricEvaluation::IO::GZip::GZip (
            const  GZip & other)  [delete]
```

Copy constructor (disabled).

Disabled because **Properties** (p. 674) member of parent cannot be copied.

Parameters

other	**GZip** (p. 465) to copy.

H.68.3 Member Function Documentation

H.68.3.1 compress() [1/6]

```
Memory::uint8Array BiometricEvaluation::IO::GZip::compress (
            const  Memory::uint8Array & uncompressedData) const  [virtual]
```

Compress a buffer.

Parameters

uncompressedData	Uncompressed data buffer to compress.

Returns

Compressed buffer.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 362).

H.68.3.2 compress() [2/6]

```
void BiometricEvaluation::IO::GZip::compress (
            const  Memory::uint8Array & uncompressedData,
            const std::string & outputFile) const  [virtual]
```

Compress a buffer.

Parameters

uncompressedData	Uncompressed data buffer to compress.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists (p. 637)	Output file already exists.
Error::StrategyError (p. 789)	**Error** (p. 112) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 362).

H.68.3.3 compress() [3/6]

```
Memory::uint8Array BiometricEvaluation::IO::GZip::compress (
            const std::string & inputFile) const  [virtual]
```

Compress a file.

Parameters

inputFile	Path to file to compress.

Returns

Compressed buffer.

Exceptions

Error::ObjectDoesNotExist (p. 637)	Input file does not exist.
Error::StrategyError (p. 789)	**Error** (p. 112) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 363).

H.68.3.4 compress() [4/6]

```
void BiometricEvaluation::IO::GZip::compress (
            const std::string & inputFile,
            const std::string & outputFile) const  [virtual]
```

Compress a file.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	Input file does not exist.
Error::ObjectExists (p. 637)	Output file already exists.
Error::StrategyError (p. 789)	**Error** (p. 112) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 364).

H.68.3.5 compress() [5/6]

```
Memory::uint8Array BiometricEvaluation::IO::GZip::compress (
            const uint8_t *const uncompressedData,
            uint64_t uncompressedDataSize) const  [virtual]
```

Compress a buffer.

Parameters

uncompressedData	Uncompressed data buffer to compress.
uncompressedDataSize	Size of uncompressed↩ Data.

Returns

Compressed buffer.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 365).

H.68.3.6 compress() [6/6]

```
void BiometricEvaluation::IO::GZip::compress (
            const uint8_t *const uncompressedData,
            uint64_t uncompressedDataSize,
            const std::string & outputFile) const  [virtual]
```

Compress a buffer.

Parameters

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

Exceptions

Error::ObjectExists *(p. 637)*	Output file already exists.
Error::StrategyError *(p. 789)*	**Error** (p. 112) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 366).

H.68.3.7 decompress() [1/6]

```
Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (
            const  Memory::uint8Array & compressedData) const  [virtual]
```

Decompress a compressed buffer.

Parameters

compressedData	Compressed data buffer to decompress.

Returns

Decompressed data.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 367).

H.68.3.8 decompress() [2/6]

```
void BiometricEvaluation::IO::GZip::decompress (
            const  Memory::uint8Array & compressedData,
            const std::string & outputFile) const  [virtual]
```

Decompress a file.

Parameters

compressedData	Compressed data buffer to decompress.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists *(p. 637)*	Output file already exists.

Error::StrategyError (p. 789)	**Error** (p. 112) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 367).

H.68.3.9 decompress() [3/6]

```
Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (
            const std::string & inputFile) const  [virtual]
```

Decompress a compressed buffer into a file.

Parameters

inputFile	Location to save compressed file.

Returns

Decompressed data.

Exceptions

Error::StrategyError (p. 789)	**Error** (p. 112) in decompression unit.
Error::ObjectDoesNotExists	Output file already exists.

Implements **BiometricEvaluation::IO::Compressor** (p. 368).

H.68.3.10 decompress() [4/6]

```
void BiometricEvaluation::IO::GZip::decompress (
            const std::string & inputFile,
            const std::string & outputFile) const  [virtual]
```

Decompress a file.

Parameters

inputFile	Path to file to decompress.

Parameters

outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectDoesNotExist (p. 637)	Input file does not exist.
Error::ObjectExists (p. 637)	Output file already exists.
Error::StrategyError (p. 789)	**Error** (p. 112) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 369).

H.68.3.11 decompress() [5/6]

```
void BiometricEvaluation::IO::GZip::decompress (
            const uint8_t *const compressedData,
            const uint64_t compressedDataSize,
            const std::string & outputFile) const  [virtual]
```

Decompress a file.

Parameters

compressedData	Compressed data buffer to decompress.
compressedDataSize	Size of compressed↩ Data.

Parameters

outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists (p. 637)	Output file already exists.
Error::StrategyError (p. 789)	**Error** (p. 112) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 369).

H.68.3.12 decompress() [6/6]

```
Memory::uint8Array BiometricEvaluation::IO::GZip::decompress (
        const uint8_t *const compressedData,
        uint64_t compressedDataSize) const  [virtual]
```

Decompress a compressed buffer.

Parameters

compressedData	Compressed data buffer to decompress.
compressedDataSize	Size of compressed↩ Data.

Returns

Decompressed data.

Exceptions

Error::StrategyError (p. 789)	**Error** (p. 112) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 370).

H.68.3.13 operator=()

```
GZip & BiometricEvaluation::IO::GZip::operator= (
            const GZip & other) [delete]
```

Assignment overload (disabled).

Disabled because **Properties** (p. 674) member of parent cannot be assigned.

Parameters

other	**GZip** (p. 465) to assign.

Returns

lhs **GZip** (p. 465).

H.68.4 Member Data Documentation

H.68.4.1 CHUNK_SIZE

```
const std::string BiometricEvaluation::IO::GZip::CHUNK_SIZE [static]
```

How many bytes to work at a time

H.68.4.2 COMPRESSION_LEVEL

```
const std::string BiometricEvaluation::IO::GZip::COMPRESSION_LEVEL [static]
```

How thorough the compression should be

H.68.4.3 COMPRESSION_METHOD

```
const std::string BiometricEvaluation::IO::GZip::COMPRESSION_METHOD [static]
```

Which underlying method in the compressor

H.68.4.4 COMPRESSION_STRATEGY

```
const std::string BiometricEvaluation::IO::GZip::COMPRESSION_STRATEGY [static]
```

Which underlying algorithm to use

H.68.4.5 INPUT_DATA_TYPE

```
const std::string BiometricEvaluation::IO::GZip::INPUT_DATA_TYPE [static]
```

The type of data being compressed

H.68.4.6 MEMORY_LEVEL

```
const std::string BiometricEvaluation::IO::GZip::MEMORY_LEVEL [static]
```

How much memory for internal compression state

H.68.4.7 WINDOW_BITS

```
const std::string BiometricEvaluation::IO::GZip::WINDOW_BITS [static]
```

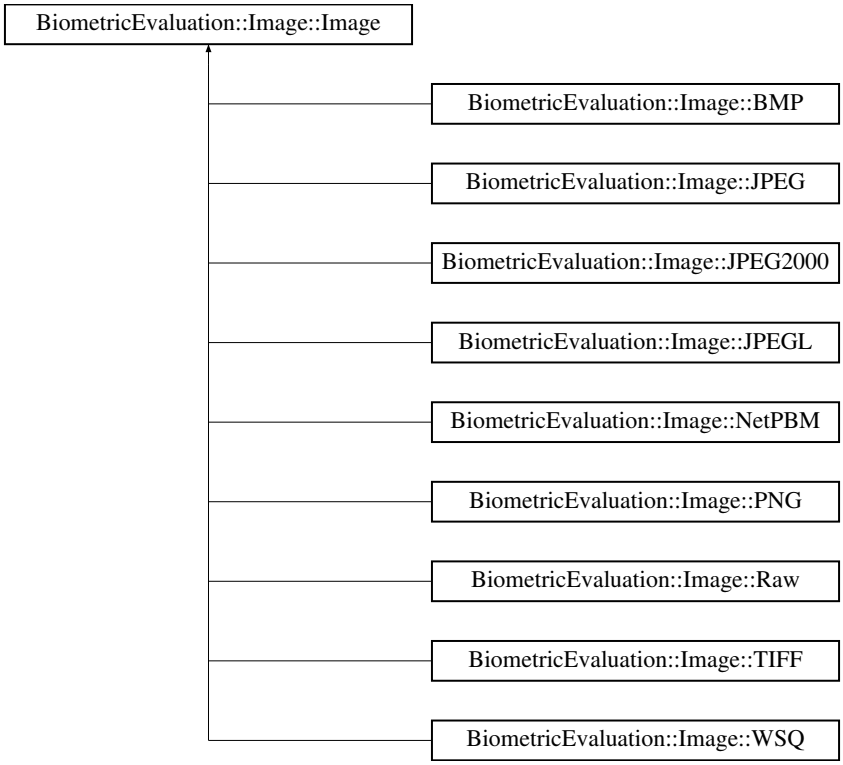
Window size

H.69 BiometricEvaluation::Image::Image Class Reference

Represent attributes common to all images.

`#include <be_image_image.h>`

Inheritance diagram for BiometricEvaluation::Image::Image:



Public Types

- using **statusCallback_t**

Public Member Functions

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm getCompressionAlgorithm** () const

  Accessor for the CompressionAlgorithm of the image.
- **Resolution getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array getData** () const

Accessor for the image data. The data returned is likely encoded in a specialized format.

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

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- virtual **Memory::uint8Array getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- virtual **Memory::uint8Array getRawGrayscaleData** (uint8_t depth) const =0

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

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

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

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

  Accessor for the presence of an alpha channel.
- statusCallback_t **getStatusCallback** () const

  Get handle to status callback function.
- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Static Public Member Functions

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.
- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of an image file and create an **Image** (p. 477) object.*
- static **CompressionAlgorithm getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.
- static **BiometricEvaluation::Image::Raw getRawImage** (const std::shared_ptr< **BiometricEvaluation::Image::Image** > &image)

  *Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).*
- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Protected Member Functions

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .
- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.
- void **setColorDepth** (const uint32_t colorDepth)

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

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

  Mutator for the presence of an alpha channel.

H.69.1 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** (p. 561), etc. Implementations of this abstraction provide the getRawData method to convert image data to 'raw' format.

Image (p. 477) resolution is in pixels per centimeter, and the coordinate system has the origin at the upper left of the image.

H.69.2 Member Typedef Documentation

H.69.2.1 statusCallback_t

```
using BiometricEvaluation::Image::Image::statusCallback_t
```

Initial value:

```
std::function<void(
              const Framework::Status)>
```

H.69.3 Constructor & Destructor Documentation

H.69.3.1 Image() [1/2]

```
BiometricEvaluation::Image::Image::Image (
             const uint8_t * data,
             const uint64_t size,
             const  Size dimensions,
             const uint32_t colorDepth,
             const uint16_t bitDepth,
             const  Resolution resolution,
             const  CompressionAlgorithm compression,
             const bool hasAlphaChannel,
             const std::string & identifier = "",
             const statusCallback_t & statusCallback =  Image::defaultStatusCallback)
```

Parent constructor for all **Image** (p. 477) classes.

Parameters

in	*data*	The image data.
in	*size*	The size of the image data, in bytes.
in	*dimensions*	The width and height of the image in pixels.
in	*colorDepth*	The image color depth, in bits-per-pixel.
in	*bitDepth*	The number of bits per color component.
in	*resolution*	The resolution of the image
in	*compression*	The Compression↩ Algorithm of data.

Parameters

in	*hasAlphaChannel*	Presence of an alpha channel.
	identifier	Identifier for the encapsulated data.
	statusCallback	Function to handle statuses sent when processing images.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) manipulating data.
Error::StrategyError *(p. 789)*	**Error** (p. 112) while creating **Image** (p. 477).

H.69.3.2 Image() [2/2]

```
BiometricEvaluation::Image::Image::Image (
            const uint8_t * data,
            const uint64_t size,
            const  CompressionAlgorithm compression,
            const std::string & identifier = "",
            const statusCallback_t & statusCallback =  Image::defaultStatusCallback)
```

Parent constructor for all **Image** (p. 477) classes.

Parameters

in	*data*	The image data.

Parameters

in	*size*	The size of the image data, in bytes.
in	*compression*	The Compression↩ Algorithm of data.
	identifier	Identifier for the encapsulated data.
	statusCallback	Function to handle statuses sent when processing images.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) manipulating data.
Error::StrategyError (p. 789)	**Error** (p. 112) while creating **Image** (p. 477).

H.69.4 Member Function Documentation

H.69.4.1 defaultStatusCallback()

```
static void BiometricEvaluation::Image::Image::defaultStatusCallback (
            const  Framework::Status & status)  [static]
```

Default handling of statuses sent from image processing libraries.

Parameters

status	Status received.

Exceptions

Error::StrategyError *(p. 789)*	status.type == **Framework::Status::Type::Error** (p. **??**)

Note

Custom implementations of signature statusCallback_t should throw an exception when status.type == **Framework::Status::Type::Error** (p. **??**).

H.69.4.2 getBitDepth()

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

Accessor for the number of bits per color component.

Returns

The bit depth of the image (in bits).

H.69.4.3 getColorDepth()

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

Accessor for the color depth of the image in bits.

Returns

The color depth of the image (bit).

H.69.4.4 getCompressionAlgorithm() [1/4]

`CompressionAlgorithm` `BiometricEvaluation::Image::Image::getCompressionAlgorithm () const`

Accessor for the CompressionAlgorithm of the image.

Returns

Type of compression used on the data that will be returned from **getData()** (p. 485).

H.69.4.5 getCompressionAlgorithm() [2/4]

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

Determine the compression algorithm of a buffer of image data.

Parameters

in	*data*	The image data.

Returns

Compression algorithm used in the buffer.

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation **Framework** (p. 124) is found.

H.69.4.6 getCompressionAlgorithm() [3/4]

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

Determine the compression algorithm of a file.

Parameters

in	*path*	Path to file.

Returns

Compression algorithm used in the file.

Exceptions

Error::ObjectDoesNotExist (p. 637)	path does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation **Framework** (p. 124) is found.

H.69.4.7 getCompressionAlgorithm() [4/4]

```
static CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm (
            const uint8_t * data,
            const uint64_t size) [static]
```

Determine the compression algorithm of a buffer of image data.

Parameters

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

Returns

Compression algorithm used in the buffer.

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation **Framework** (p. 124) is found.

H.69.4.8 getData()

`Memory::uint8Array BiometricEvaluation::Image::Image::getData () const`

Accessor for the image data. The data returned is likely encoded in a specialized format.

Returns

AutoArray holding image data.

H.69.4.9 getDataPointer()

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

Returns

Const pointer to buffer underlying _data.

H.69.4.10 getDataSize()

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

Returns

Size (p. 763) of _data.

H.69.4.11 getDimensions()

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

Accessor for the dimensions of the image in pixels.

Returns

Coordinate (p. 377) object containing dimensions in pixels.

H.69.4.12 getIdentifier()

`std::string BiometricEvaluation::Image::Image::getIdentifier () const`

Obtain the assigned image identifier.

Returns

Image (p. 477) identifier.

H.69.4.13 getRawData() [1/2]

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

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

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.

Implemented in **BiometricEvaluation::Image::BMP** (p. 332), **BiometricEvaluation::Image::JPEG2000** (p. 568), **BiometricEvaluation::Image::JPEG** (p. 563), **BiometricEvaluation::Image::JPEGL** (p. 572), **BiometricEvaluation::Image::NetPBM** (p. 632), **BiometricEvaluation::Image::PNG** (p. 662), **BiometricEvaluation::Image::Raw** (p. 691), **BiometricEvaluation::Image::TIFF** (p. 806), and **BiometricEvaluation::Image::WSQ** (p. 849).

H.69.4.14 getRawData() [2/2]

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

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

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Parameters

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

Returns

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

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.
Error::ParameterError *(p. 655)*	Propagated from **Image::removeComponents** (p. 132).
Error::StrategyError *(p. 789)*	Propagated from **Image::removeComponents** (p. 132).

H.69.4.15 getRawGrayscaleData()

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

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.
Error::NotImplemented *(p. 636)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 655)*	Invalid value for depth.

Note

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

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implemented in **BiometricEvaluation::Image::BMP** (p. 332), **BiometricEvaluation::Image::JPEG2000** (p. 569), **BiometricEvaluation::Image::JPEG** (p. 563), **BiometricEvaluation::Image::JPEGL** (p. 572), **BiometricEvaluation::Image::NetPBM** (p. 633), **BiometricEvaluation::Image::PNG** (p. 663), **BiometricEvaluation::Image::Raw** (p. 691), **BiometricEvaluation::Image::TIFF** (p. 806), and **BiometricEvaluation::Image::WSQ** (p. 849).

H.69.4.16 getRawImage()

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

Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).

Parameters

in	*image*	Shared pointer to an **Image::Image** (p. 477).

Returns

Shared pointer to an **Image::Raw** (p. 688) version of `image`.

Note

If `image` is already an **Image::Raw** (p. 688), `image` is returned to avoid a copy.

H.69.4.17 getResolution()

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

Accessor for the resolution of the image.

Returns

Resolution (p. 737) struct

H.69.4.18 getStatusCallback()

```
statusCallback_t BiometricEvaluation::Image::Image::getStatusCallback () const
```

Get handle to status callback function.

Returns

Status callback function.

H.69.4.19 hasAlphaChannel()

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

Accessor for the presence of an alpha channel.

Returns

Whether or not an alpha channel is present.

H.69.4.20 openImage() [1/3]

```
static std::shared_ptr<  Image > BiometricEvaluation::Image::Image::openImage (
            const  Memory::uint8Array & data,
            const std::string & identifier = "",
            const statusCallback_t & statusCallback =  Image::defaultStatusCallback)  [static]
```

Determine the image type of a buffer of image data and create an **Image** (p. 477) object.

Parameters

in	*data*	The image data.
	identifier	Identifier for the encapsulated data.
	statusCallback	Function to handle statuses sent when processing images.

Returns

Image (p. 477) representation of the input data buffer.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) manipulating data.
Error::StrategyError *(p. 789)*	**Error** (p. 112) while creating **Image** (p. 477).

H.69.4.21 openImage() [2/3]

```
static std::shared_ptr< Image > BiometricEvaluation::Image::Image::openImage (
            const std::string & path,
            const statusCallback_t & statusCallback = Image::defaultStatusCallback) [static]
```

Determine the image type of an image file and create an **Image** (p. 477) object.

Parameters

in	*path*	Path to image data.
	statusCallback	Function to handle statuses sent when processing images.

Returns

Image (p. 477) representation of the input data buffer.

Exceptions

Error::DataError (p. 390)	**Error** (p. 112) manipulating data.
Error::ObjectDoesNotExist (p. 637)	No file at specified path.
Error::StrategyError (p. 789)	**Error** (p. 112) while creating **Image** (p. 477).

H.69.4.22 openImage() [3/3]

```
static std::shared_ptr< Image > BiometricEvaluation::Image::Image::openImage (
            const uint8_t * data,
            const uint64_t size,
            const std::string & identifier = "",
            const statusCallback_t & statusCallback = Image::defaultStatusCallback) [static]
```

Determine the image type of a buffer of image data and create an **Image** (p. 477) object.

Parameters

in	*data*	The image data.

Parameters

in	*size*	The size of the image data, in bytes.
	identifier	Identifier for the encapsulated data.
	statusCallback	Function to handle statuses sent when processing images.

Returns

Image (p. 477) representation of the input data buffer.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) manipulating data.
Error::StrategyError *(p. 789)*	**Error** (p. 112) while creating **Image** (p. 477).

H.69.4.23 setBitDepth()

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

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

Parameters

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

H.69.4.24 setColorDepth()

```
void BiometricEvaluation::Image::Image::setColorDepth (
            const uint32_t colorDepth)  [protected]
```

Mutator for the color depth of the image in bits.

Parameters

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

H.69.4.25 setDimensions()

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

Mutator for the dimensions of the image in pixels.

Parameters

in	*dimensions*	Dimensions of image (pixel).

H.69.4.26 setHasAlphaChannel()

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

Mutator for the presence of an alpha channel.

Parameters

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

H.69.4.27 setResolution()

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

Mutator for the resolution of the image .

Parameters

in	*resolution*	**Resolution** (p. 737) struct.

H.69.4.28 valueInColorspace()

```
static uint64_t BiometricEvaluation::Image::Image::valueInColorspace (
            uint64_t color,
            uint64_t maxColorValue,
            uint8_t depth)  [static]
```

Calculate an equivalent color value for a color in an alternate colorspace.

Parameters

color	Value for color in original colorspace.
maxColorValue	Maximum value for colors in original colorspace.

Parameters

depth	Desired bit-depth of the new col-orspace.

Returns

A value equivalent to color in depth-bit space.

H.70 BiometricEvaluation::Feature::AN2K11EFS::ImageInfo Struct Reference

A structure representing information about the image and extended feature set region.

`#include <be_feature_an2k11efs.h>`

Public Attributes

- **BiometricEvaluation::Image::ROI roi**
- **FPPPosition fpp**
- **Orientation ort**
- bool **has_trv**
- TonalReversal **trv**
- bool **has_plr**
- LateralReversal **plr**

H.70.1 Detailed Description

A structure representing information about the image and extended feature set region.

H.70.2 Member Data Documentation

H.70.2.1 fpp

FPPPosition BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::fpp

The Finger/Palm/Plantar Position: Mandatory field.

H.70.2.2 ort

Orientation BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::ort

The image orientation. Optional but always present due to default value.

H.70.2.3 plr

LateralReversal BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::plr

The possible latent reversal information. Optional.

H.70.2.4 roi

`BiometricEvaluation::Image::ROI BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::roi`

The region of interest: A mandatory field.

H.70.2.5 trv

`TonalReversal BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::trv`

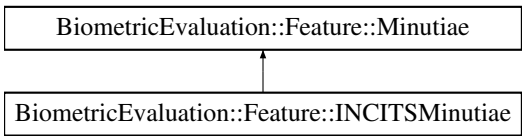
The tonal reversal information. Optional.

H.71 BiometricEvaluation::Feature::INCITSMinutiae Class Reference

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

`#include <be_feature_incitsminutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::INCITSMinutiae:



Public Member Functions

- **MinutiaeFormat getFormat** () const

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

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

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

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

  Obtains the set of delta positions. The set may be empty.
- **INCITSMinutiae** (const MinutiaPointSet &mps, const RidgeCountItemSet &rcis, const CorePointSet &cps, const DeltaPointSet &dps)

  *Construct an INCITS **Minutiae** (p. 615) object from its components.*
- **INCITSMinutiae** ()

  *Default constructor for an INCITS **Minutiae** (p. 615) object.*
- void **setMinutiaPoints** (const MinutiaPointSet &mps)

  Mutator for the minutiae point set.
- void **setRidgeCountItems** (const RidgeCountItemSet &rcis)

  Mutator for the ridge count items.
- void **setCorePointSet** (const CorePointSet &cps)

  Mutator for the set of core points.
- void **setDeltaPointSet** (const DeltaPointSet &dps)

  Mutator for the set of delta points.

Static Public Attributes

- static const std::string **FMR_ANSI_SPEC_VERSION**
- static const std::string **FMR_ISO_SPEC_VERSION**
- static const std::string **FMR_ANSI07_SPEC_VERSION**
- static const uint8_t **FMR_SPEC_VERSION_LEN** = 4
- static const uint32_t **FED_HEADER_LENGTH** = 4
- static const uint32_t **FED_RCD_ITEM_LENGTH** = 3
- static const uint16_t **FMD_MINUTIA_TYPE_MASK** = 0xC000
- static const uint16_t **FMD_RESERVED_MASK** = 0xC000
- static const uint16_t **FMD_MINUTIA_TYPE_SHIFT** = 14
- static const uint16_t **FMD_RESERVED_SHIFT** = 14
- static const uint16_t **FMD_X_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_Y_COORD_MASK** = 0x3FFF
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_MASK** = 0xC0
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_TYPE_SHIFT** = 6
- static const uint16_t **FMD_ISO_COMPACT_MINUTIA_ANGLE_MASK** = 0x3F
- static const uint16_t **FMD_MIN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MAX_MINUTIA_QUALITY** = 100
- static const uint16_t **FMD_UNKNOWN_MINUTIA_QUALITY** = 0
- static const uint16_t **FMD_MIN_MINUTIA_ANGLE** = 0
- static const uint16_t **FMD_MAX_MINUTIA_ANGLE** = 179
- static const uint16_t **FMD_MAX_MINUTIA_ISONC_ANGLE** = 255
- static const uint16_t **FMD_MAX_MINUTIA_ISOCC_ANGLE** = 63
- static const uint16_t **FMD_ANSI_ANGLE_UNIT** = 2
- static const uint16_t **FMD_ISO_ANGLE_UNIT**
- static const uint16_t **FMD_ISOCC_ANGLE_UNIT**
- static const uint16_t **FMD_MINUTIA_TYPE_OTHER** = 0
- static const uint16_t **FMD_MINUTIA_TYPE_RIDGE_ENDING** = 1
- static const uint16_t **FMD_MINUTIA_TYPE_BIFURCATION** = 2
- static const uint16_t **FMR_MIN_FINGER_QUALITY** = 0
- static const uint16_t **FMR_MAX_FINGER_QUALITY** = 100
- static const uint16_t **ISO_UNKNOWN_FINGER_QUALITY** = 0
- static const uint16_t **FED_RESERVED** = 0x0000
- static const uint16_t **FED_RIDGE_COUNT** = 0x0001
- static const uint16_t **FED_CORE_AND_DELTA** = 0x0002
- static const uint16_t **RCE_NONSPECIFIC** = 0x00
- static const uint16_t **RCE_FOUR_NEIGHBOR** = 0x01
- static const uint16_t **RCE_EIGHT_NEIGHBOR** = 0x02
- static const uint16_t **CORE_TYPE_NONANGULAR** = 0x00
- static const uint16_t **CORE_TYPE_ANGULAR** = 0x01
- static const uint16_t **DELTA_TYPE_NONANGULAR** = 0x00
- static const uint16_t **DELTA_TYPE_ANGULAR** = 0x01

H.71.1 Detailed Description

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

The base INCTISMinutiae class is responsible for reading minutiae data points and extended data. Each minutiae point, ridge count item, core, and delta is represented in the native ANSI/INCITS format. Objects of this base class cannot be instantiated, but rather derived classes are used to represent minutiae data taken from the INCITS-derived record formats.

H.71.2 Constructor & Destructor Documentation

H.71.2.1 INCITSMinutiae()

```
BiometricEvaluation::Feature::INCITSMinutiae::INCITSMinutiae (
            const MinutiaPointSet & mps,
            const RidgeCountItemSet & rcis,
            const CorePointSet & cps,
            const DeltaPointSet & dps)
```

Construct an INCITS **Minutiae** (p. 615) object from its components.

The buffer index must be set to the location in the buffer to start reading minutiae data points and extended data.

Parameters

in	*mps*	The set of minutiae points.
in	*rcis*	The set of ridge count items.
in	*cps*	The set of core points.
in	*dps*	The set of delta points.

H.71.3 Member Function Documentation

H.71.3.1 getCores()

```
CorePointSet BiometricEvaluation::Feature::INCITSMinutiae::getCores () const  [virtual]
```

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

Implements **BiometricEvaluation::Feature::Minutiae** (p. 615).

H.71.3.2 getDeltas()

```
DeltaPointSet BiometricEvaluation::Feature::INCITSMinutiae::getDeltas () const  [virtual]
```

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

Implements **BiometricEvaluation::Feature::Minutiae** (p. 615).

H.71.3.3 getFormat()

```
MinutiaeFormat BiometricEvaluation::Feature::INCITSMinutiae::getFormat () const  [virtual]
```

Obtain the minutiae format kind.

Implements **BiometricEvaluation::Feature::Minutiae** (p. 616).

H.71.3.4 getMinutiaPoints()

```
MinutiaPointSet BiometricEvaluation::Feature::INCITSMinutiae::getMinutiaPoints () const  [virtual]
```

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

Implements **BiometricEvaluation::Feature::Minutiae** (p. 616).

H.71.3.5 getRidgeCountItems()

```
RidgeCountItemSet BiometricEvaluation::Feature::INCITSMinutiae::getRidgeCountItems () const  [virtual]
```

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

Implements **BiometricEvaluation::Feature::Minutiae** (p. 616).

H.71.3.6 setCorePointSet()

```
void BiometricEvaluation::Feature::INCITSMinutiae::setCorePointSet (
            const CorePointSet & cps)
```

Mutator for the set of core points.

Parameters

in	*cps*	The set of core points.

H.71.3.7 setDeltaPointSet()

```
void BiometricEvaluation::Feature::INCITSMinutiae::setDeltaPointSet (
            const DeltaPointSet & dps)
```

Mutator for the set of delta points.

Parameters

in	*dps*	The set of delta point items.

H.71.3.8 setMinutiaPoints()

```
void BiometricEvaluation::Feature::INCITSMinutiae::setMinutiaPoints (
            const MinutiaPointSet & mps)
```

Mutator for the minutiae point set.

Parameters

in	*mps*	The minutiae points.

H.71.3.9 setRidgeCountItems()

```
void BiometricEvaluation::Feature::INCITSMinutiae::setRidgeCountItems (
            const RidgeCountItemSet & rcis)
```

Mutator for the ridge count items.

Parameters

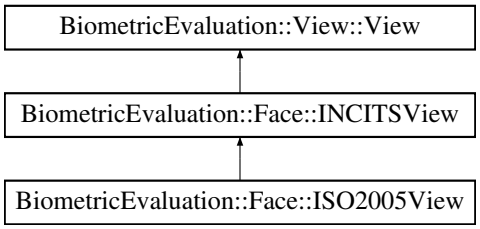
in	*rcis*	The set of ridge count items.

H.72 BiometricEvaluation::Face::INCITSView Class Reference

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

`#include <be_face_incitsview.h>`

Inheritance diagram for BiometricEvaluation::Face::INCITSView:



Public Member Functions

- **Face::Gender getGender** () const

  Obtain the gender.
- **Face::EyeColor getEyeColor** () const

  Obtain the eye color.
- **Face::HairColor getHairColor** () const

  Obtain the hair color.
- bool **propertiesConsidered** () const

  Indicate whether properties are specified.
- void **getPropertySet** (**Face::PropertySet** &propertySet) const

Get the set of properties.

- **BiometricEvaluation::Face::Expression getExpression** () const
- void **getFeaturePointSet** (BiometricEvaluation::Feature::MPEGFacePointSet &featurePointSet) const

  Obtain the set of.
- **Face::ImageType getImageType** () const

  Obtain the face image type.
- **Face::ImageDataType getImageDataType** () const

  Obtain the face image data type.
- **Face::PoseAngle getPoseAngle** () const

  Obtain the face pose angle.
- **Face::ColorSpace getColorSpace** () const

  Obtain the color space.
- **Face::SourceType getSourceType** () const

  Obtain the source type.
- uint16_t **getDeviceType** () const

  Obtain the device type.

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Protected Member Functions

- **INCITSView** (const std::string &filename, const uint32_t viewNumber)

  Construct the common components of an INCITS face view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an INCITS face view from a record contained in a buffer.
- **Memory::uint8Array** const & **getFIDData** () const

  Obtain a reference to the face image record data buffer.
- virtual void **readHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf, const uint32_t formatStandard)

  Read the common face image data record header from an INCITS record, excepting the format identifier and version number data items.
- virtual void **readFaceView** (**Memory::IndexedBuffer** &buf)

  Read the common face representation information from an INCITS record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Static Protected Attributes

- static const uint32_t **ISO2005_STANDARD** = 1
- static const uint32_t **BASE_FORMAT_ID** = 0x46414300

H.72.1 Detailed Description

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

A base **Face::INCITSView** (p. 499) class represents an INCITS/ANSI or ISO face view. This class defines the common interface for all ANSI/ISO views as well as common implementations. Subclasses specialize this class in order to represent other versions of the ANSI/ISO specs. Objects of this class cannot be created.

H.72.2 Constructor & Destructor Documentation

H.72.2.1 INCITSView() [1/2]

```
BiometricEvaluation::Face::INCITSView::INCITSView (
            const std::string & filename,
            const uint32_t viewNumber)  [protected]
```

Construct the common components of an INCITS face view from records contained in files.

See documentation in child classes of INCITS for information on constructing INCITS-derived face views.

Parameters

in	*filename*	The name of the file containing the complete face image data record.

Parameters

in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError *(p. 390)*	Invalid record format.
Error::FileError *(p. 420)*	Could not open or read from file.

H.72.2.2 INCITSView() [2/2]

```
BiometricEvaluation::Face::INCITSView::INCITSView (
            const  Memory::uint8Array & buffer,
            const uint32_t viewNumber)  [protected]
```

Construct an INCITS face view from a record contained in a buffer.

See documentation in child classes of INCITS for information on constructing INCITS-derived face views.

Parameters

in	*buffer*	The buffer containing the complete face image data record.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError *(p. 390)*	Invalid record format.

H.72.3 Member Function Documentation

H.72.3.1 getColorSpace()

Face::ColorSpace BiometricEvaluation::Face::INCITSView::getColorSpace () const

Obtain the color space.

Returns

The color space code.

H.72.3.2 getDeviceType()

uint16_t BiometricEvaluation::Face::INCITSView::getDeviceType () const

Obtain the device type.

Returns

The device type vendor code.

H.72.3.3 getEyeColor()

Face::EyeColor BiometricEvaluation::Face::INCITSView::getEyeColor () const

Obtain the eye color.

Returns

The eye color code.

H.72.3.4 getFeaturePointSet()

void BiometricEvaluation::Face::INCITSView::getFeaturePointSet (
BiometricEvaluation::Feature::MPEGFacePointSet & *featurePointSet*) const

Obtain the set of.

Parameters

out	*featurePointSet*	The set of feature points.

H.72.3.5 getFIDData()

Memory::uint8Array const & BiometricEvaluation::Face::INCITSView::getFIDData () const [protected]

Obtain a reference to the face image record data buffer.

Returns

The entire face image record data.

H.72.3.6 getGender()

Face::Gender BiometricEvaluation::Face::INCITSView::getGender () const

Obtain the gender.

Returns

The gender code.

H.72.3.7 getHairColor()

Face::HairColor BiometricEvaluation::Face::INCITSView::getHairColor () const

Obtain the hair color.

Returns

The hair color code.

H.72.3.8 getImageDataType()

Face::ImageDataType BiometricEvaluation::Face::INCITSView::getImageDataType () const

Obtain the face image data type.

Returns

The image data type.

H.72.3.9 getImageType()

Face::ImageType BiometricEvaluation::Face::INCITSView::getImageType () const

Obtain the face image type.

Returns

The image type.

H.72.3.10 getPoseAngle()

Face::PoseAngle BiometricEvaluation::Face::INCITSView::getPoseAngle () const

Obtain the face pose angle.

Returns

The pose angle.

H.72.3.11 getPropertySet()

void BiometricEvaluation::Face::INCITSView::getPropertySet (
Face::PropertySet & *propertySet*) const

Get the set of properties.

Returns

The set of properties.

H.72.3.12 getSourceType()

```
Face::SourceType BiometricEvaluation::Face::INCITSView::getSourceType () const
```

Obtain the source type.

Returns

The source type code.

H.72.3.13 propertiesConsidered()

```
bool BiometricEvaluation::Face::INCITSView::propertiesConsidered () const
```

Indicate whether properties are specified.

Returns

true if properties are specified, false otherwise.

H.72.3.14 readFaceView()

```
virtual void BiometricEvaluation::Face::INCITSView::readFaceView (
            Memory::IndexedBuffer & buf) [protected], [virtual]
```

Read the common face representation information from an INCITS record.

An **Face** (p. 113) representation from an INCITS record includes image information, gender, pose angle, etc.

Parameters

in,out	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the Facial information record.

Exceptions

DataError	The INCITS record has invalid or missing data.

H.72.3.15 readHeader()

```
virtual void BiometricEvaluation::Face::INCITSView::readHeader (
              BiometricEvaluation::Memory::IndexedBuffer & buf,
             const uint32_t formatStandard)  [protected], [virtual]
```

Read the common face image data record header from an INCITS record, excepting the format identifier and version number data items.

Parameters

in	*buf*	The indexed buffer containing the record data, with the index starting at the first octet after the format identifier and version number data items. The index of the buffer will be changed to the location after the header.

Parameters

in	*formatStandard*	Value indicating which header version to read; must be ISO2005↩ _↩ STANDARD

Exceptions

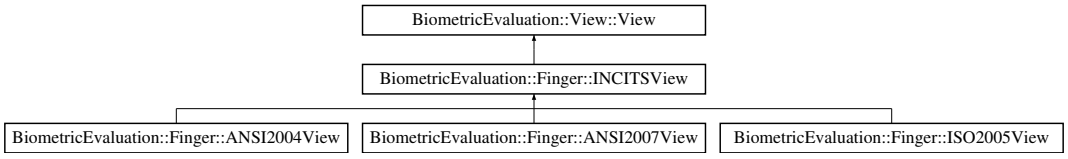
ParameterError	The formatStandard parameter is incorrect.
DataError	The INCITS record has invalid or missing data.

H.73 BiometricEvaluation::Finger::INCITSView Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_incitsview.h>`

Inheritance diagram for BiometricEvaluation::Finger::INCITSView:



Public Member Functions

- **Feature::INCITSMinutiae getMinutiaeData** () const

  Obtain the set of minutiae records.
- **Finger::Position getPosition** () const

  Obtain the finger position.
- **Finger::Impression getImpressionType** () const

  Obtain the finger impression code.
- uint32_t **getQuality** () const

  Obtain the finger quality value.
- uint16_t **getCaptureEquipmentID** () const

  Obtain the capture equipment identifier.
- bool **isAppendixFCompliant** () const

Obtain the capture equipment compliance indicator for 'Appendix F'.

- uint16_t **getProductIDOwner** () const

  Obtain the CBEFF product identifier owner.
- uint16_t **getProductIDType** () const

  Obtain the CBEFF product identifier type.
- uint32_t **getRecordLength** () const
- uint8_t **getNumFingerViews** () const
- uint8_t **getFMRReservedByte** () const
- uint32_t **getViewNumber** () const
- uint16_t **getEDBLength** () const
- std::vector< uint8_t > **getMinutiaeReservedData** () const
- void **setMinutiaeData** (const **Feature::INCITSMinutiae** &fmd)

  *Mutator for the **Feature::INCITSMinutiae*** (p. 495) *item.*
- void **setMinutiaeReservedData** (const std::vector< uint8_t > &reservedBits)

  Mutator for the FMD reserved bits vector.

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size** **getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm** **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** **convertPosition** (int incitsFGP)

  Convert a finger postion code from an INCITS finger record to the common code.
- static **Finger::Impression** **convertImpression** (int incitsIMP)

  Convert a impression type code from an INCITS finger record to the common code.

Protected Member Functions

- **INCITSView** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber)

  Construct the common components of an INCITS finger view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &fmrBuffer, const **Memory::uint8Array** &firBuffer, const uint32_t viewNumber)

  Construct an INCITS finger view from records contained in buffers.
- **Memory::uint8Array** const & **getFMRData** () const

Obtain a reference to the finger minutiae record data buffer.

- **Memory::uint8Array** const & **getFIRData** () const

  Obtain a reference to the finger image record data buffer.
- void **setPosition** (const **Finger::Position** &position)

  Mutator for the position.
- void **setImpressionType** (const **Finger::Impression** &impression)

  Mutator for the impression type.
- void **setQuality** (uint32_t quality)

  Mutator for the finger quality value.
- void **setViewNumber** (uint32_t viewNumber)

  Mutator for the finger view number.
- void **setCaptureEquipmentID** (uint16_t id)

  Mutator for the equipment ID.
- void **setCBEFFProductIDs** (uint16_t owner, uint16_t type)

  Mutator for the CBEFF Product ID owner and type.
- void **setAppendixFCompliance** (bool flag)

  Mutator for the Appendix F compliance indicator.
- void **readFMRHeader** (**Memory::IndexedBuffer** &buf, const uint32_t formatStandard)

  Read the common finger minutiae record header from an INCITS record.
- void **readFVMR** (**Memory::IndexedBuffer** &buf)

  Read the common finger view record information from an INCITS record.
- virtual std::tuple< Feature::MinutiaPointSet, std::vector< uint8_t > > **readMinutiaeDataPoints** (**Memory::IndexedBuffer** &buf, uint32_t count)

  Read the minutiae data points, and extended data blocks.
- virtual void **readExtendedDataBlock** (**Memory::IndexedBuffer** &buf)

  Read the common extended data block.
- virtual Feature::RidgeCountItemSet **readRidgeCountData** (**Memory::IndexedBuffer** &buf, uint32_t dataLength)

  Read the ridge count data.
- virtual void **readCoreDeltaData** (**Memory::IndexedBuffer** &buf, uint32_t dataLength, Feature::CorePointSet &cores, Feature::DeltaPointSet &deltas)=0

  Read the core points data.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Static Protected Attributes

- static const uint32_t **FMR_BASE_FORMAT_ID** = 0x464D5200
- static const uint32_t **ANSI2004_STANDARD** = 1

  The type of record that will be read by the subclass.
- static const uint32_t **ISO2005_STANDARD** = 2
- static const uint32_t **ANSI2007_STANDARD** = 3

H.73.1 Detailed Description

A class to represent single finger view and derived information.

A base **Finger::INCITSView** (p. 508) object represents an INCITS/ANSI or ISO finger view. This class defines the common interface for all ANSI/ISO views as well as common implementations. Subclasses specialize this class in order to represent other versions of the ANSI/ISO specs. Objects of this class cannot be created.

H.73.2 Constructor & Destructor Documentation

H.73.2.1 INCITSView() [1/2]

```
BiometricEvaluation::Finger::INCITSView::INCITSView (
            const std::string & fmrFilename,
            const std::string & firFilename,
            const uint32_t viewNumber)  [protected]
```

Construct the common components of an INCITS finger view from records contained in files.

See documentation in child classes of INCITS for information on constructing INCITS-derived finger views.

Parameters

in	*fmrFilename*	The name of the file containing the complete finger minutiae record.

Parameters

in	*firFilename*	The name of the file containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError (p. 390)	Invalid record format.
Error::FileError (p. 420)	Could not open or read from file.

H.73.2.2 INCITSView() [2/2]

```
BiometricEvaluation::Finger::INCITSView::INCITSView (
            const  Memory::uint8Array & fmrBuffer,
            const  Memory::uint8Array & firBuffer,
            const uint32_t viewNumber)  [protected]
```

Construct an INCITS finger view from records contained in buffers.

See documentation in child classes of INCITS for information on constructing INCITS-derived finger views.

Parameters

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

Parameters

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

Exceptions

Error::DataError *(p. 390)*	Invalid record format.

H.73.3 Member Function Documentation

H.73.3.1 convertImpression()

```
static Finger::Impression BiometricEvaluation::Finger::INCITSView::convertImpression (
            int incitsIMP) [static]
```

Convert a impression type code from an INCITS finger record to the common code.

Parameters

in	*incitsIMP*	A finger impression type code as defined by the INCITS standard.

Exceptions

Error::DataError *(p. 390)*	The impression type code is invalid.

Returns

The finger impression type code in common notation.

H.73.3.2 convertPosition()

```
static Finger::Position BiometricEvaluation::Finger::INCITSView::convertPosition (
            int incitsFGP) [static]
```

Convert a finger postion code from an INCITS finger record to the common code.

Parameters

in	*incitsFGP*	A finger position code as defined by the INCITS standard.

Exceptions

Error::DataError *(p. 390)*	The position code is invalid.

Returns

The finger position code in common notation.

H.73.3.3 getCaptureEquipmentID()

```
uint16_t BiometricEvaluation::Finger::INCITSView::getCaptureEquipmentID () const
```

Obtain the capture equipment identifier.

Returns

The equipment ID.

H.73.3.4 getEDBLength()

```
uint16_t BiometricEvaluation::Finger::INCITSView::getEDBLength () const
```

Returns

Length of extended data block, as recorded in the record.

H.73.3.5 getFIRData()

```
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.

H.73.3.6 getFMRData()

```
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.

H.73.3.7 getFMRReservedByte()

```
uint8_t BiometricEvaluation::Finger::INCITSView::getFMRReservedByte () const
```

Returns

Reserved byte from FMR header.

H.73.3.8 getImpressionType()

```
Finger::Impression BiometricEvaluation::Finger::INCITSView::getImpressionType () const
```

Obtain the finger impression code.

Returns

The finger impression code.

H.73.3.9 getMinutiaeReservedData()

```
std::vector< uint8_t > BiometricEvaluation::Finger::INCITSView::getMinutiaeReservedData () const
```

Returns

FMD reserved bits.

Note

Only lowest 2 bits are relevant.

H.73.3.10 getNumFingerViews()

```
uint8_t BiometricEvaluation::Finger::INCITSView::getNumFingerViews () const
```

Returns

Number of finger views, as recorded in the record.

H.73.3.11 getPosition()

```
Finger::Position BiometricEvaluation::Finger::INCITSView::getPosition () const
```

Obtain the finger position.

Returns

The finger position.

H.73.3.12 getProductIDOwner()

```
uint16_t BiometricEvaluation::Finger::INCITSView::getProductIDOwner () const [inline]
```

Obtain the CBEFF product identifier owner.

Returns

CBEFF product identifier owner.

H.73.3.13 getProductIDType()

```
uint16_t BiometricEvaluation::Finger::INCITSView::getProductIDType () const [inline]
```

Obtain the CBEFF product identifier type.

Returns

CBEFF product identifier type.

H.73.3.14 getQuality()

```
uint32_t BiometricEvaluation::Finger::INCITSView::getQuality () const
```

Obtain the finger quality value.

Returns

The finger quality value.

H.73.3.15 getRecordLength()

```
uint32_t BiometricEvaluation::Finger::INCITSView::getRecordLength () const
```

Returns

Length of record, as recorded in the record.

H.73.3.16 getViewNumber()

```
uint32_t BiometricEvaluation::Finger::INCITSView::getViewNumber () const
```

Returns

View (p. 188) number, as recorded in the record.

H.73.3.17 isAppendixFCompliant()

```
bool BiometricEvaluation::Finger::INCITSView::isAppendixFCompliant () const  [inline]
```

Obtain the capture equipment compliance indicator for 'Appendix F'.

Returns

True if 'Appendix F' compliant, false otherwise.

H.73.3.18 readCoreDeltaData()

```
virtual void BiometricEvaluation::Finger::INCITSView::readCoreDeltaData (
             Memory::IndexedBuffer & buf,
            uint32_t dataLength,
            Feature::CorePointSet & cores,
            Feature::DeltaPointSet & deltas)  [protected], [pure virtual]
```

Read the core points data.

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

Parameters

in,out	*buf*	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last core point data item.
out	*cores*	The set of core data items.
out	*deltas*	The set of delta data items.
in	*dataLength*	The length of the entire ridge count data block.

Implemented in **BiometricEvaluation::Finger::ANSI2004View** (p. 272), **BiometricEvaluation::Finger↩ ::ANSI2007View** (p. 279), and **BiometricEvaluation::Finger::ISO2005View** (p. 555).

H.73.3.19 readExtendedDataBlock()

```
virtual void BiometricEvaluation::Finger::INCITSView::readExtendedDataBlock (
            Memory::IndexedBuffer & buf)  [protected], [virtual]
```

Read the common extended data block.

Parameters

in,out	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the extended data block.

Exceptions

DataError	The INCITS record has invalid or missing data.

H.73.3.20 readFMRHeader()

```
void BiometricEvaluation::Finger::INCITSView::readFMRHeader (
            Memory::IndexedBuffer & buf,
          const uint32_t formatStandard)  [protected]
```

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

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

Parameters

in	*buf*	The indexed buffer containing the record data. The index must start after the Format ID and spec version fields in the header. The index of the buffer will be changed to the location after the header.

Parameters

in	*formatStandard*	Value indicating which header version to read; one of ANSI2004_STANDARD or ISO2005_STANDARD.

Exceptions

ParameterError	The specVersion parameter is incorrect.
DataError	The INCITS record has invalid or missing data.

H.73.3.21 readFVMR()

```
void BiometricEvaluation::Finger::INCITSView::readFVMR (
                Memory::IndexedBuffer & buf)  [protected]
```

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

A **Finger** (p. 122) **View** (p. 188) 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.

H.73.3.22 readMinutiaeDataPoints()

```
virtual std::tuple< Feature::MinutiaPointSet, std::vector< uint8_t > > BiometricEvaluation↩
::Finger::INCITSView::readMinutiaeDataPoints (
            Memory::IndexedBuffer & buf,
            uint32_t count)  [protected], [virtual]
```

Read the minutiae data points, and extended data blocks.

Function to be implemented by derived classes to read the minutiae data points and extended data block according to the specifc standard they represent.

Parameters

in	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the finger view, including the extended data.
in	*count*	Number of minutiae data points to read.

Exceptions

DataError	The INCITS record has invalid or missing data.

H.73.3.23 readRidgeCountData()

```
virtual Feature::RidgeCountItemSet BiometricEvaluation::Finger::INCITSView::readRidgeCount↩
Data (
                Memory::IndexedBuffer & buf,
              uint32_t dataLength)  [protected], [virtual]
```

Read the ridge count data.

This method reads data in the base INCITS format as defined in INCITS/ANSI 378-2004. This method may be overridden by derived classes to read data in a different record format.

Parameters

in,out	*buf*	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last ridge count item.
in	*dataLength*	The length of the entire ridge count data block.

H.73.3.24 setAppendixFCompliance()

```
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.

H.73.3.25 setCaptureEquipmentID()

```
void BiometricEvaluation::Finger::INCITSView::setCaptureEquipmentID (
            uint16_t id) [protected]
```

Mutator for the equipment ID.

Parameters

in	*id*	The equipment ID value.

H.73.3.26 setCBEFFProductIDs()

```
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.

Parameters

in	*type*	The CBEFF ID of the product type.

H.73.3.27 setImpressionType()

```
void BiometricEvaluation::Finger::INCITSView::setImpressionType (
            const  Finger::Impression & impression)  [protected]
```

Mutator for the impression type.

Parameters

in	*impression*	The finger impression type code.

H.73.3.28 setMinutiaeData()

```
void BiometricEvaluation::Finger::INCITSView::setMinutiaeData (
            const  Feature::INCITSMinutiae & fmd)
```

Mutator for the **Feature::INCITSMinutiae** (p. 495) item.

Parameters

in	*fmd*	The minutiae data object.

H.73.3.29 setMinutiaeReservedData()

```
void BiometricEvaluation::Finger::INCITSView::setMinutiaeReservedData (
            const std::vector< uint8_t > & reservedBits)
```

Mutator for the FMD reserved bits vector.

Parameters

in	*reservedBits*	Reserved bits from FMD.

H.73.3.30 setPosition()

```
void BiometricEvaluation::Finger::INCITSView::setPosition (
            const  Finger::Position & position)  [protected]
```

Mutator for the position.

Parameters

in	*position*	The finger position.

H.73.3.31 setQuality()

```
void BiometricEvaluation::Finger::INCITSView::setQuality (
            uint32_t quality)  [protected]
```

Mutator for the finger quality value.

Parameters

in	*quality*	The quality value.

H.73.3.32 setViewNumber()

```
void BiometricEvaluation::Finger::INCITSView::setViewNumber (
            uint32_t viewNumber)  [protected]
```

Mutator for the finger view number.

Parameters

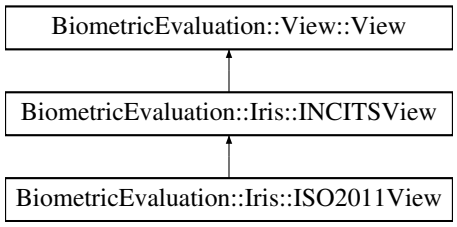
in	*viewNumber*	The view number value.

H.74 BiometricEvaluation::Iris::INCITSView Class Reference

A class to represent single iris view and derived information.

`#include <be_iris_incitsview.h>`

Inheritance diagram for BiometricEvaluation::Iris::INCITSView:



Classes

- struct **QualitySubBlock**

  Representation of an iris quality block.

Public Types

- typedef std::vector< **QualitySubBlock** > **QualitySet**

Public Member Functions

- uint8_t **getCertificationFlag** () const

  Obtain the certification flag.
- std::string **getCaptureDateString** () const

  Obtain the capture date as a string.
- **Iris::CaptureDeviceTechnology** **getCaptureDeviceTechnology** () const

  Obtain the capture device technology.
- uint16_t **getCaptureDeviceVendor** () const

  Obtain the capture device vendor.
- uint16_t **getCaptureDeviceType** () const

  Obtain the capture device type.
- void **getQualitySet** (Iris::INCITSView::QualitySet &qualitySet) const

  Obtain the set of quality sub-blocks.
- **Iris::EyeLabel** **getEyeLabel** () const

  Obtain the eye label type.
- **Iris::ImageType** **getImageType** () const

  Obtain the iris image type.
- void **getImageProperties** (**BiometricEvaluation::Iris::Orientation** &horizontalOrientation, **BiometricEvaluation::Iris::Orientation** &verticalOrientation, **BiometricEvaluation::Iris::ImageCompression** &compressionHistory) const

  Obtain the iris image properties.
- uint16_t **getCameraRange** ()

  Obtain the camera range.
- void **getRollAngleInfo** (uint16_t &rollAngle, uint16_t &rollAngleUncertainty)

Obtain the roll angle information.

- void **getIrisCenterInfo** (uint16_t &irisCenterSmallestX, uint16_t &irisCenterSmallestY, uint16_t &irisCenterLargestX, uint16_t &irisCenterLargestY, uint16_t &irisDiameterSmallest, uint16_t &irisDiameterLargest)

  Obtain the iris center information. COORDINATE_UNDEF may be returned for any of the out parameters.

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Static Public Attributes

- static const uint16_t **RANGE_UNASSIGNED** = 0
- static const uint16_t **RANGE_FAILED** = 1
- static const uint16_t **RANGE_OVERFLOW** = 65535
- static const uint16_t **ROLL_ANGLE_UNDEF** = 65535
- static const uint16_t **ROLL_UNCERTAIN_UNDEF** = 65535
- static const uint16_t **COORDINATE_UNDEF** = 0

Protected Member Functions

- **INCITSView** (const std::string &filename, const uint32_t viewNumber)

  Construct the common components of an INCITS iris view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an INCITS iris view from a record contained in a buffer.
- **Memory::uint8Array** const & **getIIRData** () const

  Obtain a reference to the iris image record data buffer.
- virtual void **readHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf, const uint32_t formatStandard)

  Read the common iris image record header from an INCITS record, excepting the format identifier and version number data items.
- virtual void **readIrisView** (**Memory::IndexedBuffer** &buf)

  Read the common iris representation information from an INCITS record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Static Protected Attributes

- static const uint32_t **ISO2011_STANDARD** = 1
- static const uint32_t **BASE_FORMAT_ID** = 0x49495200
- static const uint8_t **CAPTURE_DATE_LENGTH** = 9

H.74.1 Detailed Description

A class to represent single iris view and derived information.

A base **Iris::INCITSView** (p. 528) class represents an INCITS/ANSI or ISO iris view. This class defines the common interface for all ANSI/ISO views as well as common implementations. Subclasses specialize this class in order to represent other versions of the ANSI/ISO specs. Objects of this class cannot be created.

H.74.2 Constructor & Destructor Documentation

H.74.2.1 INCITSView() [1/2]

```
BiometricEvaluation::Iris::INCITSView::INCITSView (
            const std::string & filename,
            const uint32_t viewNumber)  [protected]
```

Construct the common components of an INCITS iris view from records contained in files.

See documentation in child classes of INCITS for information on constructing INCITS-derived iris views.

Parameters

in	*filename*	The name of the file containing the complete iris image record.

Parameters

in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError *(p. 390)*	Invalid record format.
Error::FileError *(p. 420)*	Could not open or read from file.

H.74.2.2 INCITSView() [2/2]

```
BiometricEvaluation::Iris::INCITSView::INCITSView (
            const  Memory::uint8Array & buffer,
            const uint32_t viewNumber)  [protected]
```

Construct an INCITS iris view from a record contained in a buffer.

See documentation in child classes of INCITS for information on constructing INCITS-derived iris views.

Parameters

in	*buffer*	The buffer containing the complete iris image record.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError *(p. 390)*	Invalid record format.

H.74.3 Member Function Documentation

H.74.3.1 getCameraRange()

`uint16_t BiometricEvaluation::Iris::INCITSView::getCameraRange ()`

Obtain the camera range.
RANGE_UNASSIGNED, RANGE_FAILED, or RANGE_OVERFLOW may be returned.

Returns

The camera range.

H.74.3.2 getCaptureDateString()

`std::string BiometricEvaluation::Iris::INCITSView::getCaptureDateString () const`

Obtain the capture date as a string.

Returns

The capture data and time.

H.74.3.3 getCaptureDeviceTechnology()

`Iris::CaptureDeviceTechnology` `BiometricEvaluation::Iris::INCITSView::getCaptureDeviceTechnology () const`

Obtain the capture device technology.

Returns

The capture device technology identifer.

H.74.3.4 getCaptureDeviceType()

`uint16_t BiometricEvaluation::Iris::INCITSView::getCaptureDeviceType () const`

Obtain the capture device type.

Returns

The capture device type ID.

H.74.3.5 getCaptureDeviceVendor()

`uint16_t BiometricEvaluation::Iris::INCITSView::getCaptureDeviceVendor () const`

Obtain the capture device vendor.

Returns

The capture device vendor ID.

H.74.3.6 getCertificationFlag()

`uint8_t BiometricEvaluation::Iris::INCITSView::getCertificationFlag () const`

Obtain the certification flag.

Returns

The certification flag.

H.74.3.7 getEyeLabel()

```
Iris::EyeLabel BiometricEvaluation::Iris::INCITSView::getEyeLabel () const
```

Obtain the eye label type.

Returns

The eye label.

H.74.3.8 getIIRData()

```
Memory::uint8Array const & BiometricEvaluation::Iris::INCITSView::getIIRData () const [protected]
```

Obtain a reference to the iris image record data buffer.

Returns

The entire iris image record data.

H.74.3.9 getImageProperties()

```
void BiometricEvaluation::Iris::INCITSView::getImageProperties (
            BiometricEvaluation::Iris::Orientation & horizontalOrientation,
            BiometricEvaluation::Iris::Orientation & verticalOrientation,
            BiometricEvaluation::Iris::ImageCompression & compressionHistory) const
```

Obtain the iris image properties.

Parameters

out	*horizontalOrientation*	The horizontal orientation.
out	*verticalOrientation*	The vertical orientation.
out	*compressionHistory*	The image compression history.

H.74.3.10 getImageType()

```
Iris::ImageType BiometricEvaluation::Iris::INCITSView::getImageType () const
```

Obtain the iris image type.

Returns

The image type.

H.74.3.11 getIrisCenterInfo()

```
void BiometricEvaluation::Iris::INCITSView::getIrisCenterInfo (
            uint16_t & irisCenterSmallestX,
            uint16_t & irisCenterSmallestY,
            uint16_t & irisCenterLargestX,
            uint16_t & irisCenterLargestY,
            uint16_t & irisDiameterSmallest,
            uint16_t & irisDiameterLargest)
```

Obtain the iris center information. COORDINATE_UNDEF may be returned for any of the out parameters.

Parameters

out	*irisCenterSmallestX*	Smallest expected iris center X coordinate in pixels.
out	*irisCenterSmallestY*	Smallest expected iris center Y coordinate in pixels.
out	*irisCenterLargestX*	Largest expected iris center X coordinate in pixels.

Parameters

out	*irisCenterLargestY*	Largest expected iris center Y coordinate in pixels.
out	*irisDiameterSmallest*	Smallest expected iris diameter in pixels.
out	*irisDiameterLargest*	Largest expected iris diameter in pixels.

H.74.3.12 getQualitySet()

```
void BiometricEvaluation::Iris::INCITSView::getQualitySet (
            Iris::INCITSView::QualitySet & qualitySet) const
```

Obtain the set of quality sub-blocks.

Parameters

out	*qualitySet*	The set of quality sub-blocks.

H.74.3.13 getRollAngleInfo()

```
void BiometricEvaluation::Iris::INCITSView::getRollAngleInfo (
            uint16_t & rollAngle,
            uint16_t & rollAngleUncertainty)
```

Obtain the roll angle information.

Parameters

out	*rollAngle*	The roll angle.
out	*rollAngleUncertainty*	The roll angle uncertainty.

H.74.3.14 readHeader()

```
virtual void BiometricEvaluation::Iris::INCITSView::readHeader (
              BiometricEvaluation::Memory::IndexedBuffer & buf,
            const uint32_t formatStandard)  [protected], [virtual]
```

Read the common iris image record header from an INCITS record, excepting the format identifier and version number data items.

Parameters

in	*buf*	The indexed buffer containing the record data, with the index starting at the first octet after the format identifier and version number data items. The index of the buffer will be changed to the location after the header.

Parameters

in	*formatStandard*	Value indicating which header version to read; must be ISO2011↩ _↩ STANDARD

Exceptions

ParameterError	The specVersion parameter is incorrect.
DataError	The INCITS record has invalid or missing data.

H.74.3.15 readIrisView()

```
virtual void BiometricEvaluation::Iris::INCITSView::readIrisView (
            Memory::IndexedBuffer & buf) [protected], [virtual]
```

Read the common iris representation information from an INCITS record.

An **Iris** (p. 155) Representation from an INCITS record includes image information, cropping information, etc.

Parameters

in,out	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the **Iris** (p. 155) Representation.

Exceptions

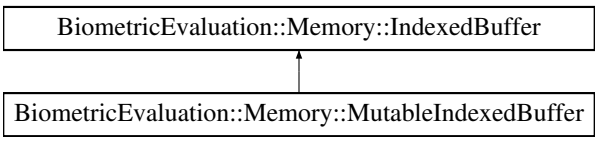
DataError	The INCITS record has invalid or missing data.

H.75 BiometricEvaluation::Memory::IndexedBuffer Class Reference

Wrap a memory buffer with an index.

`#include <be_memory_indexedbuffer.h>`

Inheritance diagram for BiometricEvaluation::Memory::IndexedBuffer:



Public Member Functions

- **IndexedBuffer** ()
- **IndexedBuffer** (const uint8_t *data, uint64_t size)

Wrap an existing buffer of a given length.

- **IndexedBuffer** (const **uint8Array** &aa)

  Wrap an existing uint8Array.
- **IndexedBuffer** (const **IndexedBuffer** &copy)=default
- uint32_t **getSize** () const

  Obtain the current size of the buffer.
- uint32_t **getIndex** () const

  Obtain the current index into the buffer.
- void **setIndex** (uint64_t index)

  Set the current index into the buffer.
- uint8_t **scanU8Val** ()

  Obtain the next element of the buffer and increment the current index value.
- uint16_t **scanU16Val** ()

  Obtain the next two elements of the buffer and increment the current index value.
- uint16_t **scanBeU16Val** ()

  Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.
- uint32_t **scanU32Val** ()

  Obtain the next four elements of the buffer and increment the current index value by four.
- uint32_t **scanBeU32Val** ()

  Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.
- uint64_t **scanU64Val** ()

  Obtain the next eight elements of the buffer and increment the current index value by eight.
- uint64_t **scan** (void *buf, uint64_t len)

  Obtain the next 'n' elements of the buffer and increment the current index value by n.
- virtual const uint8_t * **get** () const

  Returns a pointer to the managed buffer.
- virtual ~**IndexedBuffer** ()=default

H.75.1 Detailed Description

Wrap a memory buffer with an index.

The memory buffer is treated as an array of unsigned eight bit values. This class provides safe access to the array with methods to retrieve 8/16/32/64-bit elements, or and arbitrary segment starting at the index, from the array while advancing the current index. An exception is thrown by these methods whenever the retrieval would reach beyond the size of the buffer. IndexedBuffers do not own the memory of the buffers they wrap.

H.75.2 Constructor & Destructor Documentation

H.75.2.1 IndexedBuffer() [1/4]

```
BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer ()
```

Wrap a nullptr buffer.

H.75.2.2 IndexedBuffer() [2/4]

```
BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (
            const uint8_t * data,
            uint64_t size)
```

Wrap an existing buffer of a given length.

Parameters

data	Buffer to wrap.
size	Size of buffer.

H.75.2.3 IndexedBuffer() [3/4]

```
BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (
            const  uint8Array & aa)
```

Wrap an existing uint8Array.

Parameters

aa	uint8Array to wrap.

H.75.2.4 IndexedBuffer() [4/4]

```
BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (
            const  IndexedBuffer & copy)  [default]
```

Copy constructor (default).

H.75.2.5 ~IndexedBuffer()

```
virtual BiometricEvaluation::Memory::IndexedBuffer::~IndexedBuffer ()  [virtual], [default]
```

Destructor (default).

H.75.3 Member Function Documentation

H.75.3.1 get()

```
virtual const uint8_t * BiometricEvaluation::Memory::IndexedBuffer::get () const  [virtual]
```

Returns a pointer to the managed buffer.

Returns

Pointer to the managed buffer.

Reimplemented in **BiometricEvaluation::Memory::MutableIndexedBuffer** (p. 622).

H.75.3.2 getIndex()

```
uint32_t BiometricEvaluation::Memory::IndexedBuffer::getIndex () const
```

Obtain the current index into the buffer.

Returns

The current buffer index.

Note

When **getIndex()** (p. 541) == **getSize()** (p. 542), the buffer is exhausted from scanning.

H.75.3.3 getSize()

```
uint32_t BiometricEvaluation::Memory::IndexedBuffer::getSize () const
```

Obtain the current size of the buffer.

Returns

The current buffer size.

H.75.3.4 scan()

```
uint64_t BiometricEvaluation::Memory::IndexedBuffer::scan (
            void * buf,
            uint64_t len)
```

Obtain the next 'n' elements of the buffer and increment the current index value by n.

Parameters

in	*buf*	Buffer to store the copied data, or nullptr.
in	*len*	The number of elements to copy.

Exceptions

Error::DataError *(p. 390)*	The buffer is exhausted.

Returns

The number of elements copied.

H.75.3.5 scanBeU16Val()

`uint16_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU16Val ()`

Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.

Returns

The next element of the buffer as an unsigned 16-bit value.

Exceptions

Error::DataError *(p. 390)*	The buffer is exhausted.

H.75.3.6 scanBeU32Val()

`uint32_t BiometricEvaluation::Memory::IndexedBuffer::scanBeU32Val ()`

Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.

Returns

The next element of the buffer as an unsigned 32-bit value.

Exceptions

Error::DataError *(p. 390)*	The buffer is exhausted.

H.75.3.7 scanU16Val()

`uint16_t BiometricEvaluation::Memory::IndexedBuffer::scanU16Val ()`

Obtain the next two elements of the buffer and increment the current index value.

Returns

The next element of the buffer as an unsigned 16-bit value.

Exceptions

Error::DataError *(p. 390)*	The buffer is exhausted.

H.75.3.8 scanU32Val()

`uint32_t BiometricEvaluation::Memory::IndexedBuffer::scanU32Val ()`

Obtain the next four elements of the buffer and increment the current index value by four.

Returns

The next element of the buffer as an unsigned 32-bit value.

Exceptions

Error::DataError *(p. 390)*	The buffer is exhausted.

H.75.3.9 scanU64Val()

`uint64_t BiometricEvaluation::Memory::IndexedBuffer::scanU64Val ()`

Obtain the next eight elements of the buffer and increment the current index value by eight.

Returns

The next element of the buffer as an unsigned 64-bit value.

Exceptions

Error::DataError *(p. 390)*	The buffer is exhausted.

H.75.3.10 scanU8Val()

`uint8_t BiometricEvaluation::Memory::IndexedBuffer::scanU8Val ()`

Obtain the next element of the buffer and increment the current index value.

Returns

The next element of the buffer as an unsigned 8-bit value.

Exceptions

Error::DataError *(p. 390)*	The buffer is exhausted.

H.75.3.11 setIndex()

```
void BiometricEvaluation::Memory::IndexedBuffer::setIndex (
            uint64_t index)
```

Set the current index into the buffer.

Parameters

`in`	*index*	The index value to set.

Exceptions

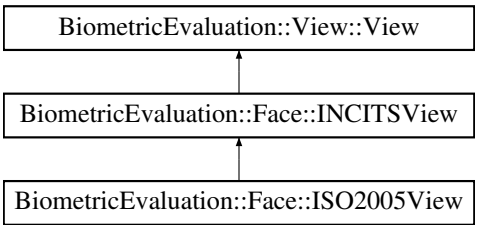
Error::ParameterError *(p. 655)*	The index parameter is too large.

H.76 BiometricEvaluation::Face::ISO2005View Class Reference

A class to represent single face view and derived information.

`#include <be_face_iso2005view.h>`

Inheritance diagram for BiometricEvaluation::Face::ISO2005View:



Public Member Functions

- **ISO2005View** ()

  Construct an empty ISO2005 ***Face*** (p. *113*) ***Image*** (p. *128*) *Data record.*
- **ISO2005View** (const std::string &filename, const uint32_t viewNumber)

  Construct an ISO 2005 face view from the named file.
- **ISO2005View** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an ISO 2005 face view from a record contained in a buffer.

Public Member Functions inherited from BiometricEvaluation::Face::INCITSView

- **Face::Gender getGender** () const

  Obtain the gender.
- **Face::EyeColor getEyeColor** () const

  Obtain the eye color.
- **Face::HairColor getHairColor** () const

  Obtain the hair color.
- bool **propertiesConsidered** () const

  Indicate whether properties are specified.
- void **getPropertySet** (**Face::PropertySet** &propertySet) const

  Get the set of properties.
- **BiometricEvaluation::Face::Expression getExpression** () const
- void **getFeaturePointSet** (BiometricEvaluation::Feature::MPEGFacePointSet &featurePointSet) const

  Obtain the set of.
- **Face::ImageType getImageType** () const

  Obtain the face image type.
- **Face::ImageDataType getImageDataType** () const

  Obtain the face image data type.
- **Face::PoseAngle getPoseAngle** () const

  Obtain the face pose angle.
- **Face::ColorSpace getColorSpace** () const

  Obtain the color space.
- **Face::SourceType getSourceType** () const

Obtain the source type.

- uint16_t **getDeviceType** () const

  Obtain the device type.

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size** **getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm** **getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Protected Member Functions

- void **readISOHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf)

  Read the face image data record header from an ISO 2005 record.

Protected Member Functions inherited from BiometricEvaluation::Face::INCITSView

- **INCITSView** (const std::string &filename, const uint32_t viewNumber)

  Construct the common components of an INCITS face view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an INCITS face view from a record contained in a buffer.
- **Memory::uint8Array** const & **getFIDData** () const

  Obtain a reference to the face image record data buffer.
- virtual void **readHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf, const uint32_t formatStandard)

  Read the common face image data record header from an INCITS record, excepting the format identifier and version number data items.
- virtual void **readFaceView** (**Memory::IndexedBuffer** &buf)

  Read the common face representation information from an INCITS record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

Mutator for the image resolution.

- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x30313000

Static Protected Attributes inherited from BiometricEvaluation::Face::INCITSView

- static const uint32_t **ISO2005_STANDARD** = 1
- static const uint32_t **BASE_FORMAT_ID** = 0x46414300

H.76.1 Detailed Description

A class to represent single face view and derived information.

A base **Face::ISO2005View** (p. 545) class represents an ISO 2005 face image data view.

H.76.2 Constructor & Destructor Documentation

H.76.2.1 ISO2005View() [1/2]

```
BiometricEvaluation::Face::ISO2005View::ISO2005View (
            const std::string & filename,
            const uint32_t viewNumber)
```

Construct an ISO 2005 face view from the named file.

The entire face image data record is passed into this method, with the specific instance of the facial image that is to be extraced from the record.

Parameters

in	*filename*	The name of the file containing the complete face image data record.

Parameters

in	*viewNumber*	The facial information instance to read.

Exceptions

Error::DataError (p. 390)	Invalid record format.
Error::FileError (p. 420)	Could not open or read from file.

H.76.2.2 ISO2005View() [2/2]

```
BiometricEvaluation::Face::ISO2005View::ISO2005View (
            const  Memory::uint8Array & buffer,
            const uint32_t viewNumber)
```

Construct an ISO 2005 face view from a record contained in a buffer.

The entire face image data record is passed into this method, with the specific instance of the facial image that is to be extraced from the record.

Parameters

in	*buffer*	The buffer containing the complete face image data record.
in	*viewNumber*	The facial information instance to read.

Exceptions

Error::DataError *(p. 390)*	Invalid record format.

H.76.3 Member Function Documentation

H.76.3.1 readISOHeader()

```
void BiometricEvaluation::Face::ISO2005View::readISOHeader (
            BiometricEvaluation::Memory::IndexedBuffer & buf)  [protected]
```

Read the face image data record header from an ISO 2005 record.

Parameters

in	*buf*	The indexed buffer containing the record data. The index of the buffer will be changed to the location after the header.

Exceptions

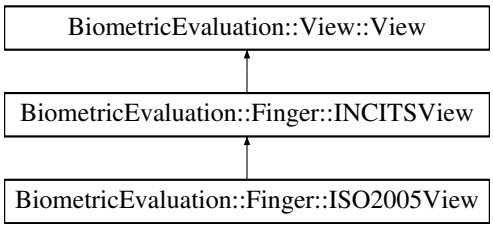
DataError	The record has invalid or missing data.

H.77 BiometricEvaluation::Finger::ISO2005View Class Reference

A class to represent single finger view and derived information.

`#include <be_finger_iso2005view.h>`

Inheritance diagram for BiometricEvaluation::Finger::ISO2005View:



Public Member Functions

- **ISO2005View** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t viewNumber)

  Construct an ISO-2005 finger view from records contained in files.
- **ISO2005View** (const **Memory::uint8Array** &fmrBuffer, const **Memory::uint8Array** &firBuffer, const uint32_t viewNumber)

  Construct an ISO-2005 finger view from records contained in buffers.

Public Member Functions inherited from BiometricEvaluation::Finger::INCITSView

- **Feature::INCITSMinutiae getMinutiaeData** () const

  Obtain the set of minutiae records.
- **Finger::Position getPosition** () const

  Obtain the finger position.
- **Finger::Impression getImpressionType** () const

  Obtain the finger impression code.
- uint32_t **getQuality** () const

  Obtain the finger quality value.
- uint16_t **getCaptureEquipmentID** () const

  Obtain the capture equipment identifier.
- bool **isAppendixFCompliant** () const

  Obtain the capture equipment compliance indicator for 'Appendix F'.
- uint16_t **getProductIDOwner** () const

  Obtain the CBEFF product identifier owner.
- uint16_t **getProductIDType** () const

  Obtain the CBEFF product identifier type.
- uint32_t **getRecordLength** () const
- uint8_t **getNumFingerViews** () const
- uint8_t **getFMRReservedByte** () const
- uint32_t **getViewNumber** () const
- uint16_t **getEDBLength** () const
- std::vector< uint8_t > **getMinutiaeReservedData** () const
- void **setMinutiaeData** (const **Feature::INCITSMinutiae** &fmd)

  *Mutator for the **Feature::INCITSMinutiae** (p. 495) item.*
- void **setMinutiaeReservedData** (const std::vector< uint8_t > &reservedBits)

  Mutator for the FMD reserved bits vector.

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size** **getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm** **getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Protected Member Functions

- void **readFMRHeader** (**Memory::IndexedBuffer** &buf)
- void **readCoreDeltaData** (**Memory::IndexedBuffer** &buf, uint32_t dataLength, Feature::CorePoint↩ Set &cores, Feature::DeltaPointSet &deltas)

  Read the core points data.

Protected Member Functions inherited from BiometricEvaluation::Finger::INCITSView

- **INCITSView** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view↩ Number)

  Construct the common components of an INCITS finger view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &fmrBuffer, const **Memory::uint8Array** &firBuffer, const uint32_t viewNumber)

  Construct an INCITS finger view from records contained in buffers.
- **Memory::uint8Array** const & **getFMRData** () const

  Obtain a reference to the finger minutiae record data buffer.
- **Memory::uint8Array** const & **getFIRData** () const

  Obtain a reference to the finger image record data buffer.
- void **setPosition** (const **Finger::Position** &position)

  Mutator for the position.
- void **setImpressionType** (const **Finger::Impression** &impression)

  Mutator for the impression type.
- void **setQuality** (uint32_t quality)

  Mutator for the finger quality value.
- void **setViewNumber** (uint32_t viewNumber)

  Mutator for the finger view number.
- void **setCaptureEquipmentID** (uint16_t id)

  Mutator for the equipment ID.
- void **setCBEFFProductIDs** (uint16_t owner, uint16_t type)

  Mutator for the CBEFF Product ID owner and type.

- void **setAppendixFCompliance** (bool flag)

  Mutator for the Appendix F compliance indicator.
- void **readFMRHeader** (**Memory::IndexedBuffer** &buf, const uint32_t formatStandard)

  Read the common finger minutiae record header from an INCITS record.
- void **readFVMR** (**Memory::IndexedBuffer** &buf)

  Read the common finger view record information from an INCITS record.
- virtual std::tuple< Feature::MinutiaPointSet, std::vector< uint8_t > > **readMinutiaeDataPoints** (**Memory::IndexedBuffer** &buf, uint32_t count)

  Read the minutiae data points, and extended data blocks.
- virtual void **readExtendedDataBlock** (**Memory::IndexedBuffer** &buf)

  Read the common extended data block.
- virtual Feature::RidgeCountItemSet **readRidgeCountData** (**Memory::IndexedBuffer** &buf, uint32_t dataLength)

  Read the ridge count data.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x20323000

Static Protected Attributes inherited from BiometricEvaluation::Finger::INCITSView

- static const uint32_t **FMR_BASE_FORMAT_ID** = 0x464D5200
- static const uint32_t **ANSI2004_STANDARD** = 1

  The type of record that will be read by the subclass.
- static const uint32_t **ISO2005_STANDARD** = 2
- static const uint32_t **ANSI2007_STANDARD** = 3

Additional Inherited Members

Static Public Member Functions inherited from BiometricEvaluation::Finger::INCITSView

- static **Finger::Position convertPosition** (int incitsFGP)

  Convert a finger postion code from an INCITS finger record to the common code.
- static **Finger::Impression convertImpression** (int incitsIMP)

  Convert a impression type code from an INCITS finger record to the common code.

H.77.1 Detailed Description

A class to represent single finger view and derived information.

A **Finger::ISO2005View** (p. 549) object represents a finger view from a ISO/IEC-2005 **Finger** (p. 122) Minutiae Record.

H.77.2 Constructor & Destructor Documentation

H.77.2.1 ISO2005View() [1/2]

```
BiometricEvaluation::Finger::ISO2005View::ISO2005View (
            const std::string & fmrFilename,
            const std::string & firFilename,
            const uint32_t viewNumber)
```

Construct an ISO-2005 finger view from records contained in files.

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

Parameters

in	*fmrFilename*	The name of the file containing the complete finger minutiae record.

Parameters

in	*firFilename*	The name of the file containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

H.77.2.2 ISO2005View() [2/2]

```
BiometricEvaluation::Finger::ISO2005View::ISO2005View (
            const  Memory::uint8Array & fmrBuffer,
            const  Memory::uint8Array & firBuffer,
            const uint32_t viewNumber)
```

Construct an ISO-2005 finger view from records contained in buffers.

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

Parameters

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

Parameters

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

Exceptions

Error::DataError *(p. 390)*	Invalid record format.

H.77.3 Member Function Documentation

H.77.3.1 readCoreDeltaData()

```
void BiometricEvaluation::Finger::ISO2005View::readCoreDeltaData (
             Memory::IndexedBuffer & buf,
            uint32_t dataLength,
            Feature::CorePointSet & cores,
            Feature::DeltaPointSet & deltas)  [protected], [virtual]
```

Read the core points data.

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

Parameters

in,out	*buf*	The indexed buffer containing the record data. On function exit, the buffer index will be set to the location after the last core point data item.
out	*cores*	The set of core data items.
out	*deltas*	The set of delta data items.
in	*dataLength*	The length of the entire ridge count data block.

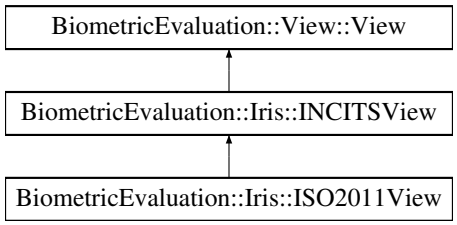
Implements **BiometricEvaluation::Finger::INCITSView** (p. 517).

H.78 BiometricEvaluation::Iris::ISO2011View Class Reference

A class to represent single iris view and derived information.

`#include <be_iris_iso2011view.h>`

Inheritance diagram for BiometricEvaluation::Iris::ISO2011View:



Public Member Functions

- **ISO2011View** ()

  Construct an empty ISO 2011 iris view.
- **ISO2011View** (const std::string &filename, const uint32_t viewNumber)

  Construct an ISO 2011 iris view from the named file.
- **ISO2011View** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an ISO 2011 iris view from a record contained in a buffer.

Public Member Functions inherited from BiometricEvaluation::Iris::INCITSView

- uint8_t **getCertificationFlag** () const

  Obtain the certification flag.
- std::string **getCaptureDateString** () const

  Obtain the capture date as a string.
- **Iris::CaptureDeviceTechnology** **getCaptureDeviceTechnology** () const

  Obtain the capture device technology.
- uint16_t **getCaptureDeviceVendor** () const

  Obtain the capture device vendor.
- uint16_t **getCaptureDeviceType** () const

  Obtain the capture device type.
- void **getQualitySet** (Iris::INCITSView::QualitySet &qualitySet) const

  Obtain the set of quality sub-blocks.
- **Iris::EyeLabel** **getEyeLabel** () const

  Obtain the eye label type.
- **Iris::ImageType** **getImageType** () const

  Obtain the iris image type.
- void **getImageProperties** (**BiometricEvaluation::Iris::Orientation** &horizontalOrientation, **Biometric↩ Evaluation::Iris::Orientation** &verticalOrientation, **BiometricEvaluation::Iris::ImageCompression** &compressionHistory) const

  Obtain the iris image properties.
- uint16_t **getCameraRange** ()

  Obtain the camera range.

- void **getRollAngleInfo** (uint16_t &rollAngle, uint16_t &rollAngleUncertainty)

  Obtain the roll angle information.
- void **getIrisCenterInfo** (uint16_t &irisCenterSmallestX, uint16_t &irisCenterSmallestY, uint16_t &irisCenterLargestX, uint16_t &irisCenterLargestY, uint16_t &irisDiameterSmallest, uint16_t &irisDiameterLargest)

  Obtain the iris center information. COORDINATE_UNDEF may be returned for any of the out parameters.

Public Member Functions inherited from BiometricEvaluation::View::View

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size** **getImageSize** () const

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

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm** **getCompressionAlgorithm** () const

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

  Obtain the image scan resolution.

Protected Member Functions

- void **readISOHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf)

Protected Member Functions inherited from BiometricEvaluation::Iris::INCITSView

- **INCITSView** (const std::string &filename, const uint32_t viewNumber)

  Construct the common components of an INCITS iris view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an INCITS iris view from a record contained in a buffer.
- **Memory::uint8Array** const & **getIIRData** () const

  Obtain a reference to the iris image record data buffer.
- virtual void **readHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf, const uint32_t formatStandard)

  Read the common iris image record header from an INCITS record, excepting the format identifier and version number data items.
- virtual void **readIrisView** (**Memory::IndexedBuffer** &buf)

  Read the common iris representation information from an INCITS record.

Protected Member Functions inherited from BiometricEvaluation::View::View

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

Mutator for the image resolution.

- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x30323000

Static Protected Attributes inherited from BiometricEvaluation::Iris::INCITSView

- static const uint32_t **ISO2011_STANDARD** = 1
- static const uint32_t **BASE_FORMAT_ID** = 0x49495200
- static const uint8_t **CAPTURE_DATE_LENGTH** = 9

Additional Inherited Members

Public Types inherited from BiometricEvaluation::Iris::INCITSView

- typedef std::vector< **QualitySubBlock** > **QualitySet**

Static Public Attributes inherited from BiometricEvaluation::Iris::INCITSView

- static const uint16_t **RANGE_UNASSIGNED** = 0
- static const uint16_t **RANGE_FAILED** = 1
- static const uint16_t **RANGE_OVERFLOW** = 65535
- static const uint16_t **ROLL_ANGLE_UNDEF** = 65535
- static const uint16_t **ROLL_UNCERTAIN_UNDEF** = 65535
- static const uint16_t **COORDINATE_UNDEF** = 0

H.78.1 Detailed Description

A class to represent single iris view and derived information.

An Iris::ISO2011VIEW class represents an ISO 19794-6 iris image record view.

H.78.2 Constructor & Destructor Documentation

H.78.2.1 ISO2011View() [1/2]

```
BiometricEvaluation::Iris::ISO2011View::ISO2011View (
            const std::string & filename,
            const uint32_t viewNumber)
```

Construct an ISO 2011 iris view from the named file.

Parameters

in	*filename*	The name of the file containing the complete iris image record.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError (p. 390)	Invalid record format.
Error::FileError (p. 420)	Could not open or read from file.

H.78.2.2 ISO2011View() [2/2]

```
BiometricEvaluation::Iris::ISO2011View::ISO2011View (
            const  Memory::uint8Array & buffer,
            const uint32_t viewNumber)
```

Construct an ISO 2011 iris view from a record contained in a buffer.

Parameters

in	*buffer*	The buffer containing the complete iris image record.
in	*viewNumber*	The eye number to use.

Exceptions

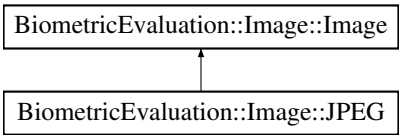
Error::DataError *(p. 390)*	Invalid record format.

H.79 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, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **JPEG** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Memory::uint8Array getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.
- **Memory::uint8Array getRawData** () const

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

Public Member Functions inherited from BiometricEvaluation::Image::Image

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm getCompressionAlgorithm** () const

  Accessor for the CompressionAlgorithm of the image.
- **Resolution getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array getData** () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual **Memory::uint8Array getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Size getDimensions** () const

Accessor for the dimensions of the image in pixels.

- uint32_t **getColorDepth** () const

  Accessor for the color depth of the image in bits.

- uint16_t **getBitDepth** () const

  Accessor for the number of bits per color component.

- bool **hasAlphaChannel** () const

  Accessor for the presence of an alpha channel.

- statusCallback_t **getStatusCallback** () const

  Get handle to status callback function.

- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Static Public Member Functions

- static bool **isJPEG** (const uint8_t *data, uint64_t size)
- static int **getc_skip_marker_segment** (const unsigned short marker, unsigned char **cbufptr, unsigned char *ebufptr)

Static Public Member Functions inherited from BiometricEvaluation::Image::Image

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.

- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*

- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*

- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of an image file and create an **Image** (p. 477) object.*

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.

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

  *Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).*

- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::Image::Image

- using **statusCallback_t**

Protected Member Functions inherited from BiometricEvaluation::Image::Image

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .
- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.
- void **setColorDepth** (const uint32_t colorDepth)

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

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

  Mutator for the presence of an alpha channel.

H.79.1 Detailed Description

A JPEG-encoded image.

H.79.2 Member Function Documentation

H.79.2.1 getRawData()

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

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

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 486).

H.79.2.2 getRawGrayscaleData()

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

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError (p. 390)	**Error** (p. 112) decompressing image data.
Error::NotImplemented (p. 636)	Unsupported conversion based on source color depth.
Error::ParameterError (p. 655)	Invalid value for depth.

Note

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

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implements **BiometricEvaluation::Image::Image** (p. 487).

H.79.2.3 isJPEG()

```
static bool BiometricEvaluation::Image::JPEG::isJPEG (
            const uint8_t * data,
            uint64_t size)  [static]
```

Whether or not data is a Lossy **JPEG** (p. 561) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

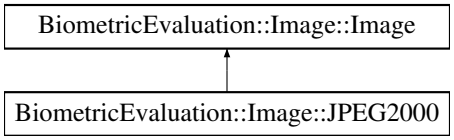
true if data appears to be a Lossy **JPEG** (p. 561) image, false otherwise

H.80 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 std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**, const int8_t codecFormat=2)

  *Create a new **JPEG2000** (p. 565) object.*
- **JPEG2000** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Memory::uint8Array getRawData** () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::uint8Array getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.

Public Member Functions inherited from BiometricEvaluation::Image::Image

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm getCompressionAlgorithm** () const

Accessor for the CompressionAlgorithm of the image.

- **Resolution getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array getData** () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual **Memory::uint8Array getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Size getDimensions** () const

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

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

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

  Accessor for the presence of an alpha channel.
- statusCallback_t **getStatusCallback** () const

  Get handle to status callback function.
- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Static Public Member Functions

- static bool **isJPEG2000** (const uint8_t *data, uint64_t size)

Static Public Member Functions inherited from BiometricEvaluation::Image::Image

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.
- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of an image file and create an **Image** (p. 477) object.*
- static **CompressionAlgorithm getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.
- static **BiometricEvaluation::Image::Raw getRawImage** (const std::shared_ptr< **BiometricEvaluation::Image::Image** > &image)

  *Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).*
- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::Image::Image

- using **statusCallback_t**

Protected Member Functions inherited from BiometricEvaluation::Image::Image

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .
- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.
- void **setColorDepth** (const uint32_t colorDepth)

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

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

  Mutator for the presence of an alpha channel.

H.80.1 Detailed Description

A JPEG-2000-encoded image.

H.80.2 Constructor & Destructor Documentation

H.80.2.1 JPEG2000()

```
BiometricEvaluation::Image::JPEG2000::JPEG2000 (
            const uint8_t * data,
            const uint64_t size,
            const std::string & identifier = "",
            const statusCallback_t & statusCallback = Image::defaultStatusCallback,
            const int8_t codecFormat = 2)
```

Create a new **JPEG2000** (p. 565) object.

Parameters

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

Parameters

	statusCallback	Function to handle statuses sent when processing images.
in	*codec*	The OPJ_CODEC_FORMAT used to encode data.

Exceptions

Error::DataError (p. 390)	**Error** (p. 112) manipulating data.
Error::StrategyError (p. 789)	**Error** (p. 112) while creating **Image** (p. 477).

H.80.3 Member Function Documentation

H.80.3.1 getRawData()

Memory::uint8Array BiometricEvaluation::Image::JPEG2000::getRawData () const [virtual]

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

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError (p. 390)	**Error** (p. 112) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 486).

H.80.3.2 getRawGrayscaleData()

```
Memory::uint8Array BiometricEvaluation::Image::JPEG2000::getRawGrayscaleData (
            uint8_t depth) const  [virtual]
```

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError (p. 390)	**Error** (p. 112) decompressing image data.
Error::NotImplemented (p. 636)	Unsupported conversion based on source color depth.
Error::ParameterError (p. 655)	Invalid value for depth.

Note

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

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implements **BiometricEvaluation::Image::Image** (p. 487).

H.80.3.3 isJPEG2000()

```
static bool BiometricEvaluation::Image::JPEG2000::isJPEG2000 (
            const uint8_t * data,
            uint64_t size)  [static]
```

Whether or not data is a JPEG-2000 image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

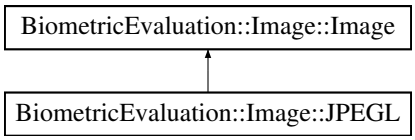
true if data appears to be a JPEG-2000 image, false otherwise.

H.81 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, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **JPEGL** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Memory::uint8Array getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.
- **Memory::uint8Array getRawData** () const

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

Public Member Functions inherited from BiometricEvaluation::Image::Image

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm getCompressionAlgorithm** () const

  Accessor for the CompressionAlgorithm of the image.

- **Resolution getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array getData** () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual **Memory::uint8Array getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Size getDimensions** () const

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

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

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

  Accessor for the presence of an alpha channel.
- statusCallback_t **getStatusCallback** () const

  Get handle to status callback function.
- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Static Public Member Functions

- static bool **isJPEGL** (const uint8_t *data, uint64_t size)

Static Public Member Functions inherited from BiometricEvaluation::Image::Image

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.
- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of an image file and create an **Image** (p. 477) object.*
- static **CompressionAlgorithm getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.
- static **BiometricEvaluation::Image::Raw getRawImage** (const std::shared_ptr< **BiometricEvaluation::Image::Image** > &image)

  *Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).*
- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::Image::Image

- using **statusCallback_t**

Protected Member Functions inherited from BiometricEvaluation::Image::Image

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .
- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.
- void **setColorDepth** (const uint32_t colorDepth)

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

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

  Mutator for the presence of an alpha channel.

H.81.1 Detailed Description

A Lossless JPEG-encoded image.

H.81.2 Member Function Documentation

H.81.2.1 getRawData()

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

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

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 486).

H.81.2.2 getRawGrayscaleData()

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

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError (p. *390*)	**Error** (p. 112) decompressing image data.
Error::NotImplemented (p. *636*)	Unsupported conversion based on source color depth.
Error::ParameterError (p. *655*)	Invalid value for depth.

Note

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

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implements **BiometricEvaluation::Image::Image** (p. 487).

H.81.2.3 isJPEGL()

```
static bool BiometricEvaluation::Image::JPEGL::isJPEGL (
            const uint8_t * data,
            uint64_t size)  [static]
```

Whether or not data is a Lossless **JPEG** (p. 561) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

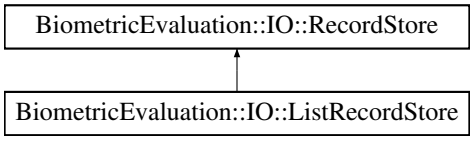
true if data appears to be a Lossless **JPEG** (p. 561) image, false otherwise.

H.82 BiometricEvaluation::IO::ListRecordStore Class Reference

IO::RecordStore (p. 700) that reads a list of keys from a text file, and retrieves the data from another **IO::RecordStore** (p. 700).

`#include <be_io_listrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ListRecordStore:



Public Member Functions

- **ListRecordStore** (const std::string &pathname)
- ~**ListRecordStore** ()=default
- void **insert** (const std::string &key, const void *const data, const uint64_t size) override
- void **remove** (const std::string &key) override
- **Memory::uint8Array read** (const std::string &key) const override

  Read a complete record from a store.
- void **replace** (const std::string &key, const void *const data, const uint64_t size) override final
- uint64_t **length** (const std::string &key) const override
- void **flush** (const std::string &key) const override
- void **sync** () const override
- **RecordStore::Record sequence** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  *Move the **RecordStore** (p. 700).*
- uint64_t **getSpaceUsed** () const override

  Obtain real storage utilization.

- unsigned int **getCount** () const override
- std::string **getPathname** () const override
- std::string **getDescription** () const override
- void **changeDescription** (const std::string &description) override
- virtual void **insert** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const **Memory::uint8Array** &data)

Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- virtual bool **containsKey** (const std::string &key) const

  *Determines whether the **RecordStore** (p. 700) contains an element with the specified key.*
- virtual **iterator** **begin** () noexcept
- virtual **iterator** **end** () noexcept

Additional Inherited Members

Public Types inherited from BiometricEvaluation::IO::RecordStore

- enum class **Kind** {
  BerkeleyDB , **Archive** , **File** , **SQLite** ,
  Compressed , **List** , **Default** = BerkeleyDB }
- using **Record** = struct Record
- using **iterator** = **IO::RecordStoreIterator**

Static Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- static bool **isRecordStore** (const std::string &pathname)

  *Determine if a location appears to be a **RecordStore** (p. 700).*
- static std::shared_ptr< **RecordStore** > **openRecordStore** (const std::string &pathname, **IO::Mode** mode= **Mode::ReadOnly**)

  *Open an existing **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static std::shared_ptr< **RecordStore** > **createRecordStore** (const std::string &pathname, const std↩ ::string &description, const **IO::RecordStore::Kind** &kind)

  *Create a new **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static void **removeRecordStore** (const std::string &pathname)
- static void **mergeRecordStores** (const std::string &mergePathname, const std::string &description, const **IO::RecordStore::Kind** &kind, const std::vector< std::string > &pathnames, const std::function< bool()> &interrupt=[ ]() {return(false);})

  *Create a new **RecordStore** (p. 700) that contains the contents of several other RecordStores.*

Static Public Attributes inherited from BiometricEvaluation::IO::RecordStore

- static const std::string **INVALIDKEYCHARS**
- static const int **BE_RECSTORE_SEQ_START** = 1
- static const int **BE_RECSTORE_SEQ_NEXT** = 2

H.82.1 Detailed Description

IO::RecordStore (p. 700) that reads a list of keys from a text file, and retrieves the data from another **IO::RecordStore** (p. 700).

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** (p. 700) 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** (p. 700); see example below.

Second, create a file called 'KeyList.txt' in the **RecordStore** (p. 700) 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 = TestLRS Type = List Source Record Store = /Users/wsalamon/sandbox/SD29.rs

Note

List RecordStores must be opened read-only.

Important

The list of keys is only consulted when iterating the **ListRecordStore** (p. 574). Read methods invoked manually will succeed for any key present in the backing **RecordStore** (p. 700), regardless of the key's presence in the explicit list of keys.

H.82.2 Constructor & Destructor Documentation

H.82.2.1 ListRecordStore()

```
BiometricEvaluation::IO::ListRecordStore::ListRecordStore (
            const std::string & pathname)
```

Constructor, always opening read-only

H.82.2.2 ~ListRecordStore()

```
BiometricEvaluation::IO::ListRecordStore::~ListRecordStore ()  [default]
```

Destructor

H.82.3 Member Function Documentation

H.82.3.1 changeDescription()

```
void BiometricEvaluation::IO::ListRecordStore::changeDescription (
            const std::string & description)  [override], [virtual]
```

Change the description of the **RecordStore** (p. 700).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 703).

H.82.3.2 flush()

```
void BiometricEvaluation::IO::ListRecordStore::flush (
            const std::string & key) const  [override], [virtual]
```

Commit the record's data to storage.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 704).

H.82.3.3 getCount()

```
unsigned int BiometricEvaluation::IO::ListRecordStore::getCount () const  [override], [virtual]
```

Obtain the number of items in the **RecordStore** (p. 700).

Returns

The number of items in the **RecordStore** (p. 700).

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.82.3.4 getDescription()

```
std::string BiometricEvaluation::IO::ListRecordStore::getDescription () const  [override], [virtual]
```

Obtain a textual description of the **RecordStore** (p. 700).

Returns

The **RecordStore** (p. 700)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.82.3.5 getPathname()

```
std::string BiometricEvaluation::IO::ListRecordStore::getPathname () const  [override], [virtual]
```

Return the path name of the **RecordStore** (p. 700).

Returns

Where in the file system the **RecordStore** (p. 700) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.82.3.6 getSpaceUsed()

```
uint64_t BiometricEvaluation::IO::ListRecordStore::getSpaceUsed () const  [override], [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** (p. 700).

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 706).

H.82.3.7 insert() [1/2]

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

Insert a record into the store.

Parameters

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

Exceptions

Error::ObjectExists *(p. 637)*	A record with the given key is already present.
Error::StrategyError *(p. 789)*	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 706).

H.82.3.8 insert() [2/2]

```
void BiometricEvaluation::IO::ListRecordStore::insert (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [override], [virtual]
```

Insert a record into the store.

Parameters

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

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Implements **BiometricEvaluation::IO::RecordStore** (p. 707).

H.82.3.9 length()

```
uint64_t BiometricEvaluation::IO::ListRecordStore::length (
            const std::string & key) const  [override], [virtual]
```

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 708).

H.82.3.10 move()

```
void BiometricEvaluation::IO::ListRecordStore::move (
            const std::string & pathname)  [override], [virtual]
```

Move the **RecordStore** (p. 700).

The **RecordStore** (p. 700) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 700).

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 710).

H.82.3.11 read()

```
Memory::uint8Array BiometricEvaluation::IO::ListRecordStore::read (
            const std::string & key) const  [override], [virtual]
```

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

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

Returns

The record associated with the key.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 712).

H.82.3.12 remove()

```
void BiometricEvaluation::IO::ListRecordStore::remove (
            const std::string & key)  [override], [virtual]
```

Remove a record from the store.

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 713).

H.82.3.13 replace() [1/2]

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

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.82.3.14 replace() [2/2]

```
void BiometricEvaluation::IO::ListRecordStore::replace (
            const std::string & key,
            const void *const data,
            const uint64_t size) [final], [override], [virtual]
```

Replace a complete record in a **RecordStore** (p. 700).

Parameters

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

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.82.3.15 sequence()

```
RecordStore::Record BiometricEvaluation::IO::ListRecordStore::sequence (
            int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the function to return the next record. The starting point is typically the first record, and is set to that when the

RecordStore (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The record that is currently in sequence.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	End of sequencing.
Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 715).

H.82.3.16 sequenceKey()

```
std::string BiometricEvaluation::IO::ListRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key.

Sequencing means to start at some point in the store and return the key, then repeatedly calling the function to return the next key. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

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

Returns

The key of the currently sequenced record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 716).

H.82.3.17 setCursorAtKey()

```
void BiometricEvaluation::IO::ListRecordStore::setCursorAtKey (
            const std::string & key)  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 700), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 582).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 582).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.82.3.18 sync()

```
void BiometricEvaluation::IO::ListRecordStore::sync () const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

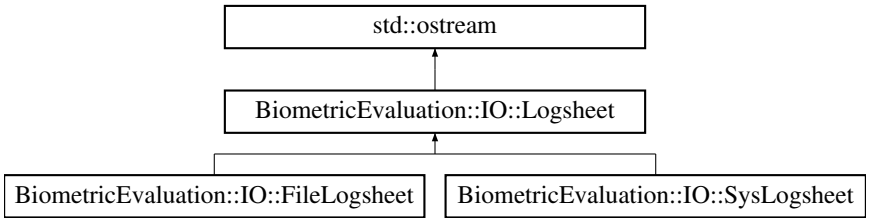
Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.83 BiometricEvaluation::IO::Logsheet Class Reference

A class to represent a logging mechanism.

`#include <be_io_logsheet.h>`

Inheritance diagram for BiometricEvaluation::IO::Logsheet:



Public Types

- enum class **Kind** { **Null** , **File** , **Syslog** }

Public Member Functions

- **Logsheet** ()

  *Create a **Logsheet** (p. 585) that has no backing store. A log entry is maintained, but cannot be permanently stored. This is the Null **Logsheet** (p. 585).*
- virtual ~**Logsheet** ()
- void **newEntry** ()

  Start a new entry, causing the existing entry to be closed and written.
- std::string **getCurrentEntry** () const

  Obtain the contents of the current entry currently under construction.
- void **resetCurrentEntry** ()
- uint32_t **getCurrentEntryNumber** () const

  Obtain the current entry number.
- virtual void **write** (const std::string &entry)

  Write a string as an entry to the backing store.
- virtual void **writeComment** (const std::string &entry)

  Write a string as a comment to the backing store.
- virtual void **writeDebug** (const std::string &entry)

  Write a string as a debug entry to the backing store.
- void **setCommit** (const bool state)

  Enable or disable the commitment of normal entries to the backing log storage.
- bool **getCommit** () const

  Get the current entry commit state.

- void **setDebugCommit** (const bool state)

  Enable or disable the commitment of debug entries to the backing log storage.
- bool **getDebugCommit** () const

  Get the current debug entry commit state.
- void **setCommentCommit** (const bool state)

  Enable or disable the commitment of comment entries to the backing log storage.
- bool **getCommentCommit** () const

  Get the current comment entry commit state.
- virtual void **sync** ()

  Synchronize any buffered data to the underlying backing store.
- void **setAutoSync** (bool state)
- bool **getAutoSync** () const

Static Public Member Functions

- static **Logsheet::Kind** **getTypeFromURL** (const std::string &url)

  *Map the URL scheme, taken from a string containing the entire URL, into a **Logsheet*** (p. 585) *type.*
- static bool **lineIsEntry** (const std::string &line)

  Helper function to determine whether a string is a valid log entry.
- static bool **lineIsComment** (const std::string &line)

  Helper function to determine whether a string is a valid comment log entry.
- static bool **lineIsDebug** (const std::string &line)

  Helper function to determine whether a string is a valid debug log entry.
- static std::string **trim** (const std::string &entry)

  *Trim delimiters from **Logsheet*** (p. 585) *entries.*

Static Public Attributes

- static const char **CommentDelimiter** = '#'
- static const char **EntryDelimiter** = 'E'
- static const char **DebugDelimiter** = 'D'
- static const std::string **DescriptionTag**
- static const std::string **FILEURLSCHEME**
- static const std::string **SYSLOGURLSCHEME**

Protected Member Functions

- void **incrementEntryNumber** ()

  Increment the current entry number.
- std::string **getCurrentEntryNumberAsString** () const

  Obtain the current entry 'tag', in 'Edddd' format.

H.83.1 Detailed Description

A class to represent a logging mechanism.

A **Logsheet** (p. 585) is an output stream, so applications can write into the stream as a staging area using the << operator, then start a new entry by calling **newEntry()** (p. 591). Entries in the log are prefixed with an entry number, which is incremented when the entry is written (either by directly calling **write()** (p. 594), or calling **newEntry()** (p. 591)).

How the log data is stored is implemented by subclasses of **Logsheet** (p. 585).

Note

By default, the entries in the **Logsheet** (p. 585) may not be immediately written to the backing store, depending on the buffering behavior of the operating system. Applications can force a write by invoking **sync()** (p. 593), 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** (p. 585) won't check for that condition, but any existing **Logsheet** (p. 585) that is re-opened for append may have an incorrect starting entry number.

H.83.2 Member Enumeration Documentation

H.83.2.1 Kind

```
enum class BiometricEvaluation::IO::Logsheet::Kind [strong]
```

Enumerator

Null	No backing store log sheet
File	File-based log sheet
Syslog	Syslog daemon backing store

H.83.3 Constructor & Destructor Documentation

H.83.3.1 ~Logsheet()

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

Destructor

H.83.4 Member Function Documentation

H.83.4.1 getAutoSync()

`bool BiometricEvaluation::IO::Logsheet::getAutoSync () const`

Return the current auto-sync state.

Returns

true if auto-sync is on, false otherwise.

H.83.4.2 getCommentCommit()

`bool BiometricEvaluation::IO::Logsheet::getCommentCommit () const`

Get the current comment entry commit state.

Returns

true if comment entries are committed to the backing store, false otherwise.

H.83.4.3 getCommit()

`bool BiometricEvaluation::IO::Logsheet::getCommit () const`

Get the current entry commit state.

Returns

true if normal entries are to be committed, false if not.

H.83.4.4 getCurrentEntry()

`std::string BiometricEvaluation::IO::Logsheet::getCurrentEntry () const`

Obtain the contents of the current entry currently under construction.

Returns

The text of the current entry.

H.83.4.5 getCurrentEntryNumber()

`uint32_t BiometricEvaluation::IO::Logsheet::getCurrentEntryNumber () const`

Obtain the current entry number.

Returns

The current entry number.

H.83.4.6 getCurrentEntryNumberAsString()

`std::string BiometricEvaluation::IO::Logsheet::getCurrentEntryNumberAsString () const [protected]`

Obtain the current entry 'tag', in 'Edddd' format.

Returns

The text of the current entry tag.

H.83.4.7 getDebugCommit()

```
bool BiometricEvaluation::IO::Logsheet::getDebugCommit () const
```

Get the current debug entry commit state.

Returns

true if debug entries are committed to the backing store, false otherwise.

H.83.4.8 getTypeFromURL()

```
static Logsheet::Kind BiometricEvaluation::IO::Logsheet::getTypeFromURL (
            const std::string & url)  [static]
```

Map the URL scheme, taken from a string containing the entire URL, into a **Logsheet** (p. 585) type.

Parameters

in	*url*	The uniform resource locator of the **Logsheet** (p. 585).

Returns

The type of **Logsheet** (p. 585) represented by the URL.

Exceptions

Error::ParameterError *(p. 655)*	The URL scheme is missing or invalid.

H.83.4.9 lineIsComment()

```
static bool BiometricEvaluation::IO::Logsheet::lineIsComment (
            const std::string & line)  [static]
```

Helper function to determine whether a string is a valid comment log entry.

Parameters

in	*line*	The string potentially containing a comment entry.

Returns

true if the string is a comment entry, false otherwise.

H.83.4.10 lineIsDebug()

```
static bool BiometricEvaluation::IO::Logsheet::lineIsDebug (
            const std::string & line)  [static]
```

Helper function to determine whether a string is a valid debug log entry.

Parameters

in	*line*	The string potentially containing a debug entry.

Returns

true if the string is a debug entry, false otherwise.

H.83.4.11 lineIsEntry()

```
static bool BiometricEvaluation::IO::Logsheet::lineIsEntry (
            const std::string & line)  [static]
```

Helper function to determine whether a string is a valid log entry.

Parameters

in	*line*	The string potentially containing a log entry.

Returns

true if the string is a log entry, false otherwise.

H.83.4.12 newEntry()

```
void BiometricEvaluation::IO::Logsheet::newEntry ()
```

Start a new entry, causing the existing entry to be closed and written.

Applications do not have to call this method for the first entry, however, as the stream is ready for writing upon construction.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying backing store.

H.83.4.13 resetCurrentEntry()

```
void BiometricEvaluation::IO::Logsheet::resetCurrentEntry ()
```

Reset the current entry buffer to the beginning.

H.83.4.14 setAutoSync()

```
void BiometricEvaluation::IO::Logsheet::setAutoSync (
            bool state)
```

Turn on/off auto-sync of the data. Applications may gain performance by turning off auto-sync, or gain reliability by turning it on.

Parameters

state	When true, the data is sync'd whenever **new↩ Entry()** (p. 591) is or **write()** (p. 594) is called. When false, **sync()** (p. 593) must be called to force a write.

H.83.4.15 setCommentCommit()

```
void BiometricEvaluation::IO::Logsheet::setCommentCommit (
            const bool state)
```

Enable or disable the commitment of comment entries to the backing log storage.

When comment entry commitment is disabled, calls to writeComment may still be made, but those entries do not appear in the log backing store.

Parameters

in	*state*	true if comment entries are to be committed, false if not.

H.83.4.16 setCommit()

```
void BiometricEvaluation::IO::Logsheet::setCommit (
            const bool state)
```

Enable or disable the commitment of normal entries to the backing log storage.

When entry commitment is disabled, the entry number is not incremented. Entries may be streamed into the object, and new entries created.

Parameters

in	*state*	True if normal entries are to be committed, false if not.

H.83.4.17 setDebugCommit()

```
void BiometricEvaluation::IO::Logsheet::setDebugCommit (
            const bool state)
```

Enable or disable the commitment of debug entries to the backing log storage.

When debug entry commitment is disabled, calls to writeDebug may still be made, but those entries do not appear in the log backing store.

Parameters

in	*state*	true if debug entries are to be committed, false if not.

H.83.4.18 sync()

```
virtual void BiometricEvaluation::IO::Logsheet::sync ()  [virtual]
```

Synchronize any buffered data to the underlying backing store.

This syncing is dependent on the behavior of the underlying storage mechanism.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying backing store.

Reimplemented in **BiometricEvaluation::IO::FileLogsheet** (p. 431), and **BiometricEvaluation::IO::SysLogsheet** (p. 800).

H.83.4.19 trim()

```
static std::string BiometricEvaluation::IO::Logsheet::trim (
            const std::string & entry)  [static]
```

Trim delimiters from **Logsheet** (p. 585) entries.

Works for comments and numbered entries.

Parameters

in	*entry*	The entry to trim.

Returns

Delimiter-less entry.

H.83.4.20 write()

```
virtual void BiometricEvaluation::IO::Logsheet::write (
            const std::string & entry)  [virtual]
```

Write a string as an entry to the backing store.

This does not affect the current log entry buffer, but does increment the entry number.

Parameters

in	*entry*	The text of the log entry.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying backing store.

Reimplemented in **BiometricEvaluation::IO::FileLogsheet** (p. 432), and **BiometricEvaluation::IO::SysLogsheet** (p. 800).

H.83.4.21 writeComment()

```
virtual void BiometricEvaluation::IO::Logsheet::writeComment (
            const std::string & entry)  [virtual]
```

Write a string as a comment to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

in	*entry*	The text of the comment.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying backing store.

Reimplemented in **BiometricEvaluation::IO::FileLogsheet** (p. 432), and **BiometricEvaluation::IO::SysLogsheet** (p. 801).

H.83.4.22 writeDebug()

```
virtual void BiometricEvaluation::IO::Logsheet::writeDebug (
            const std::string & entry)  [virtual]
```

Write a string as a debug entry to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A debug line is prefixed with DebugDelimiter followed by a space.

Parameters

in	*entry*	The text of the debug message.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when logging.

Reimplemented in **BiometricEvaluation::IO::FileLogsheet** (p. 432), and **BiometricEvaluation::IO::SysLogsheet** (p. 801).

H.83.5 Member Data Documentation

H.83.5.1 CommentDelimiter

```
const char BiometricEvaluation::IO::Logsheet::CommentDelimiter = '#'  [static]
```

Delimiter for a comment line in the log sheet.

H.83.5.2 DebugDelimiter

```
const char BiometricEvaluation::IO::Logsheet::DebugDelimiter = 'D'  [static]
```

Delimiter for an debug line in the log sheet.

H.83.5.3 DescriptionTag

```
const std::string BiometricEvaluation::IO::Logsheet::DescriptionTag  [static]
```

The tag for the description string.

H.83.5.4 EntryDelimiter

```
const char BiometricEvaluation::IO::Logsheet::EntryDelimiter = 'E'  [static]
```

Delimiter for an entry line in the log sheet.

H.83.5.5 FILEURLSCHEME

```
const std::string BiometricEvaluation::IO::Logsheet::FILEURLSCHEME  [static]
```

The URL scheme to be used for **FileLogsheet** (p. 424) URL strings.

H.83.5.6 SYSLOGURLSCHEME

```
const std::string BiometricEvaluation::IO::Logsheet::SYSLOGURLSCHEME  [static]
```

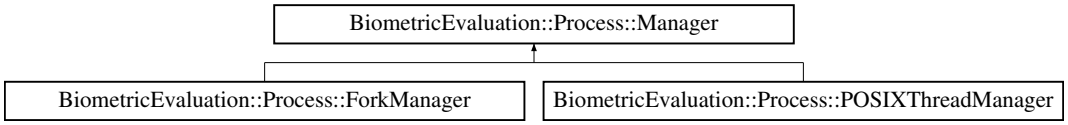
The URL scheme to be used for **SysLogsheet** (p. 793) URL strings.

H.84 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 *(p. 596) constructor.*
- virtual std::shared_ptr< **WorkerController** > **addWorker** (std::shared_ptr< **Worker** > worker)=0

  *Adds a **Worker** (p. 828) to be managed by this **Manager** (p. 596).*
- virtual uint32_t **getNumCompletedWorkers** () const

  Obtain the number of Workers that have exited.
- virtual uint32_t **getNumActiveWorkers** () const

  Obtain the number of Workers that are still working.
- virtual uint32_t **getTotalWorkers** () const

  Obtain the number of Workers this class is handling.
- virtual void **startWorkers** (bool wait=true, bool communicate=false)=0

  *Begin **Worker** (p. 828)'s work.*
- virtual void **startWorker** (std::shared_ptr< **WorkerController** > worker, bool wait=true, bool communicate=false)=0

  *Start a **Worker** (p. 828).*
- virtual void **waitForWorkerExit** ()=0

Block until all Workers have exited.

- virtual void **reset** ()

  Reuse all Workers.

- virtual void **stopWorker** (std::shared_ptr< **WorkerController** > worker)=0

  *Ask **Worker** (p. 828) to return as soon as possible.*

- virtual bool **waitForMessage** (std::shared_ptr< **WorkerController** > &sender, int *nextFD=nullptr, int numSeconds=-1) const

  *Wait for a message from a **Worker** (p. 828).*

- virtual bool **getNextMessage** (std::shared_ptr< **WorkerController** > &sender, **Memory::uint8Array** &message, int numSeconds=-1) const

  *Obtain a message from a **Worker** (p. 828).*

- virtual void **broadcastMessage** (**Memory::uint8Array** &message) const

  Send one message to all Workers.

- virtual ~**Manager** ()

  ***Manager** (p. 596) destructor.*

Protected Member Functions

- virtual void **_wait** ()=0

  Do not return until all spawned processes exited.

Protected Attributes

- std::vector< std::shared_ptr< **WorkerController** > > **_workers**
- std::vector< std::shared_ptr< **WorkerController** > > **_pendingExit**

H.84.1 Detailed Description

An interface for intranode process management classes.

H.84.2 Member Function Documentation

H.84.2.1 addWorker()

```
virtual std::shared_ptr< WorkerController > BiometricEvaluation::Process::Manager::addWorker
(
            std::shared_ptr< Worker > worker)  [pure virtual]
```

Adds a **Worker** (p. 828) to be managed by this **Manager** (p. 596).

Parameters

worker	A **Worker** (p. 828) instance to run.

Returns

shared_ptr to worker.

Implemented in **BiometricEvaluation::Process::ForkManager** (p. 449), and **BiometricEvaluation::Process::POSIXThreadManager** (p. 668).

H.84.2.2 broadcastMessage()

```
virtual void BiometricEvaluation::Process::Manager::broadcastMessage (
            Memory::uint8Array & message) const  [virtual]
```

Send one message to all Workers.

Parameters

message	The message to send to all Workers.

Exceptions

Error::StrategyError (p. 789)	**Error** (p. 112) propagated from the **WorkerController** (p. 834).

H.84.2.3 getNextMessage()

```
virtual bool BiometricEvaluation::Process::Manager::getNextMessage (
            std::shared_ptr< WorkerController > & sender,
             Memory::uint8Array & message,
            int numSeconds = -1) const  [virtual]
```

Obtain a message from a **Worker** (p. 828).

Parameters

out	*sender*	Reference to a shared pointer of the **WorkerController** (p. 834) that sent the message.

Parameters

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 (p. 637)	(Unexpected) widowed pipe.
Error::StrategyError (p. 789)	**Error** (p. 112) receiving message.

H.84.2.4 getNumActiveWorkers()

```
virtual uint32_t BiometricEvaluation::Process::Manager::getNumActiveWorkers () const  [virtual]
```

Obtain the number of Workers that are still working.

Returns

The number of Workers that are still working.

Exceptions

Error::StrategyError (p. 789)	No Workers have started working yet.

H.84.2.5 getNumCompletedWorkers()

```
virtual uint32_t BiometricEvaluation::Process::Manager::getNumCompletedWorkers () const  [virtual]
```

Obtain the number of Workers that have exited.

Returns

The number of Workers that have exited.

Exceptions

Error::StrategyError *(p. 789)*	No Workers have started working yet.

H.84.2.6 getTotalWorkers()

```
virtual uint32_t BiometricEvaluation::Process::Manager::getTotalWorkers () const  [virtual]
```

Obtain the number of Workers this class is handling.

Returns

Number of Workers.

H.84.2.7 reset()

```
virtual void BiometricEvaluation::Process::Manager::reset ()  [virtual]
```

Reuse all Workers.

Exceptions

Error::ObjectExists *(p. 637)*	At least one **Worker** (p. 828) is still working.

H.84.2.8 startWorker()

```
virtual void BiometricEvaluation::Process::Manager::startWorker (
            std::shared_ptr< WorkerController > worker,
            bool wait = true,
            bool communicate = false)  [pure virtual]
```

Start a **Worker** (p. 828).

Parameters

	worker	Pointer to a **WorkerController** (p. 834) that is being managed by this **Manager** (p. 596) instance.

Parameters

	wait	Whether or not to wait for this **Worker** (p. 828) to exit before returning control to the caller.
in	*communicate*	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists *(p. 637)*	worker is already working.
Error::StrategyError *(p. 789)*	worker is not managed by this **Manager** (p. 596) 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** (p. 453), and **BiometricEvaluation::↩ Process::POSIXThreadManager** (p. 668).

H.84.2.9 startWorkers()

```
virtual void BiometricEvaluation::Process::Manager::startWorkers (
            bool wait = true,
            bool communicate = false)  [pure virtual]
```

Begin **Worker** (p. 828)'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 (p. 637)	At least one **Worker** (p. 828) is already working.
Error::StrategyError (p. 789)	Problem starting Workers.

Implemented in **BiometricEvaluation::Process::ForkManager** (p. 455), and **BiometricEvaluation::↩ Process::POSIXThreadManager** (p. 670).

H.84.2.10 stopWorker()

```
virtual void BiometricEvaluation::Process::Manager::stopWorker (
            std::shared_ptr< WorkerController > worker) [pure virtual]
```

Ask **Worker** (p. 828) to return as soon as possible.

Parameters

worker	Pointer to the **Worker Controller** (p. 834) that should be stopped.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	worker is not working.
Error::StrategyError *(p. 789)*	Problem asking worker to stop.

Implemented in **BiometricEvaluation::Process::ForkManager** (p. 455), and **BiometricEvaluation::Process::POSIXThreadManager** (p. 670).

H.84.2.11 waitForMessage()

```
virtual bool BiometricEvaluation::Process::Manager::waitForMessage (
            std::shared_ptr< WorkerController > & sender,
            int * nextFD = nullptr,
            int numSeconds = -1) const [virtual]
```

Wait for a message from a **Worker** (p. 828).

Parameters

out	*sender*	Reference to a shared pointer of the **Worker Controller** (p. 834) that sent the message.
in,out	*nextFD*	Location to store a pipe that has data to read.

Parameters

in	*numSeconds*	Number of seconds to wait for a message, or < 0 to block.

Returns

true if there is a **Worker** (p. 828) sending a message false otherwise or if an error occurred.

H.84.2.12 waitForWorkerExit()

```
virtual void BiometricEvaluation::Process::Manager::waitForWorkerExit () [pure virtual]
```

Block until all Workers have exited.

Use this method if wait=false was set during a call to startWorker(s) but now wait=true is desired.

Implemented in **BiometricEvaluation::Process::ForkManager** (p. 456), and **BiometricEvaluation::Process::POSIXThreadManager** (p. 671).

H.84.3 Member Data Documentation

H.84.3.1 _pendingExit

```
std::vector<std::shared_ptr< WorkerController> > BiometricEvaluation::Process::Manager::_pendingExit [protected]
```

Workers that are about to exit (stop requested).

H.84.3.2 _workers

```
std::vector<std::shared_ptr< WorkerController> > BiometricEvaluation::Process::Manager::_workers [protected]
```

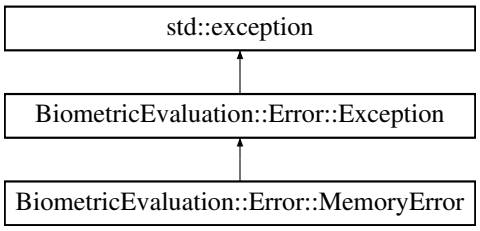
Workers that have been added.

H.85 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.85.1 Detailed Description

An error occurred when allocating an object.

H.85.2 Constructor & Destructor Documentation

H.85.2.1 MemoryError() [1/2]

```
BiometricEvaluation::Error::MemoryError::MemoryError ()
```

Construct a **MemoryError** (p. 604) object with the default information string.

H.85.2.2 MemoryError() [2/2]

```
BiometricEvaluation::Error::MemoryError::MemoryError (
            const std::string & info)
```

Construct a **MemoryError** (p. 604) object with an information string appended to the default information string.

H.86 BiometricEvaluation::System::MemoryLogger Class Reference

Public Member Functions

- **MemoryLogger** (const std::shared_ptr< **IO::Logsheet** > &logSheet)

  Begin logging memory information every `interval`, starting immediately.
- std::string **getComment** () const

  Get the comment that is appended to every auto logger entry.
- void **addLogEntry** ()

  Log memory information immediately.
- void **setComment** (std::string_view comment)

  Set a comment for each log entry.

- void **startAutoLogging** (std::chrono::microseconds interval, bool writeHeader=true)

  Begin logging memory information every `interval`*, starting immediately.*
- void **stopAutoLogging** ()

  Stop logging memory information automatically.

H.86.1 Member Function Documentation

H.86.1.1 addLogEntry()

```
void BiometricEvaluation::System::MemoryLogger::addLogEntry ()
```

Log memory information immediately.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when writing to the Logsheet.

H.86.1.2 getComment()

```
std::string BiometricEvaluation::System::MemoryLogger::getComment () const
```

Get the comment that is appended to every auto logger entry.

Returns

The comment string.

H.86.1.3 setComment()

```
void BiometricEvaluation::System::MemoryLogger::setComment (
            std::string_view comment)
```

Set a comment for each log entry.

The comment string is auto-appended to the end of each log entry.

Parameters

comment	Comment string

H.86.1.4 startAutoLogging()

```
void BiometricEvaluation::System::MemoryLogger::startAutoLogging (
            std::chrono::microseconds interval,
            bool writeHeader = true)
```

Begin logging memory information every `interval`, starting immediately.

Parameters

interval	The gap between logging snapshots.
writeHeader	Whether to write the header once (as a comment) before logging contents.

Exceptions

Error::ObjectExists (p. 637)	Already autologging.

H.86.1.5 stopAutoLogging()

`void BiometricEvaluation::System::MemoryLogger::stopAutoLogging ()`

Stop logging memory information automatically.

Exceptions

Error::ObjectDoesNotExist (p. 637)	Not currently logging.

H.87 BiometricEvaluation::Process::MessageCenter Class Reference

`#include <be_process_messagecenter.h>`

Public Member Functions

- **MessageCenter** (uint32_t port= **MessageCenter::DEFAULT_PORT**)

  Constructor.
- bool **hasUnseenMessages** () const

Determine whether or not there are unseen messages.

- bool **getNextMessage** (uint32_t &clientID, **Memory::uint8Array** &message, int numSeconds=-1)

  Get the next available message.
- void **sendResponse** (uint32_t clientID, const **Memory::uint8Array** &message) const

  Send a message to a client.
- void **disconnectClient** (uint32_t clientID)

  Break the connection with a client.

Static Public Attributes

- static const int **CONNECTION_BACKLOG** = 10
- static const uint16_t **DEFAULT_PORT** = 7899
- static const int **DEFAULT_TIMEOUT** = 1
- static const uint64_t **MAX_MESSAGE_LENGTH** = 255

H.87.1 Detailed Description

Convenience for asynchronous TCP socket message passing.

H.87.2 Constructor & Destructor Documentation

H.87.2.1 MessageCenter()

```
BiometricEvaluation::Process::MessageCenter::MessageCenter (
            uint32_t port = MessageCenter::DEFAULT_PORT)
```

Constructor.

Parameters

port	Listening port.

H.87.3 Member Function Documentation

H.87.3.1 disconnectClient()

```
void BiometricEvaluation::Process::MessageCenter::disconnectClient (
            uint32_t clientID)
```

Break the connection with a client.

Parameters

clientID	ID of the client to disconect.

H.87.3.2 getNextMessage()

```
bool BiometricEvaluation::Process::MessageCenter::getNextMessage (
            uint32_t & clientID,
             Memory::uint8Array & message,
            int numSeconds = -1)
```

Get the next available message.

Parameters

out	*clientID*	ID of the client that sent the message.
in,out	*message*	Message received.
in	*numSeconds*	Number of seconds to wait for a message, or < 0 to block indefinitely.

Returns

true if a message was received before timing out.

H.87.3.3 hasUnseenMessages()

```
bool BiometricEvaluation::Process::MessageCenter::hasUnseenMessages () const
```

Determine whether or not there are unseen messages.

Returns

true if a message has been received and not read.

Note

Returns immediately.

H.87.3.4 sendResponse()

```
void BiometricEvaluation::Process::MessageCenter::sendResponse (
            uint32_t clientID,
            const  Memory::uint8Array & message) const
```

Send a message to a client.

Parameters

clientID	ID of client to receive message.
message	Message to send client.

H.87.4 Member Data Documentation

H.87.4.1 CONNECTION_BACKLOG

```
const int BiometricEvaluation::Process::MessageCenter::CONNECTION_BACKLOG = 10  [static]
```

Number of outstanding connections.

H.87.4.2 DEFAULT_PORT

```
const uint16_t BiometricEvaluation::Process::MessageCenter::DEFAULT_PORT = 7899  [static]
```

Default port used for messages.

H.87.4.3 DEFAULT_TIMEOUT

```
const int BiometricEvaluation::Process::MessageCenter::DEFAULT_TIMEOUT = 1  [static]
```

Default number of seconds to wait between polls.

H.87.4.4 MAX_MESSAGE_LENGTH

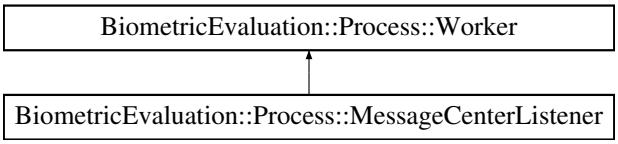
```
const uint64_t BiometricEvaluation::Process::MessageCenter::MAX_MESSAGE_LENGTH = 255  [static]
```

Maximum length of a message.

H.88 BiometricEvaluation::Process::MessageCenterListener Class Reference

```
#include <be_process_mclistener.h>
```

Inheritance diagram for BiometricEvaluation::Process::MessageCenterListener:



Public Member Functions

- int32_t **workerMain** ()

  The method that will get called to start execution by a ProcessManager.

Public Member Functions inherited from BiometricEvaluation::Process::Worker

- std::shared_ptr< void > **getParameter** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828).*
- double **getParameterAsDouble** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828) as a double.*
- int64_t **getParameterAsInteger** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828) as an integer.*
- std::string **getParameterAsString** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828) as a string.*
- void **setParameter** (const std::string &name, std::shared_ptr< void > argument)

  *Pass a parameter to this **Worker** (p. 828).*
- virtual void **stop** () final

  *Tell this **Worker** (p. 828) to return ASAP.*
- void **closeWorkerPipeEnds** ()

  *Perform initialization for communication from **Worker** (p. 828) to **Manager** (p. 596).*
- void **closeManagerPipeEnds** ()

  *Perform initialization for communication from **Manager** (p. 596) to **Worker** (p. 828).*
- int **getSendingPipe** () const

  *Obtain the pipe used to send messages to this **Worker** (p. 828).*
- int **getReceivingPipe** () const

  *Obtain the pipe used to receive messages to this **Worker** (p. 828).*
- void **sendMessageToManager** (const **Memory::uint8Array** &message)

  *Send a message to the **Manager** (p. 596).*
- void **receiveMessageFromManager** (**Memory::uint8Array** &message)

  *Receive a message from the **Manager** (p. 596).*
- void **_initCommunication** ()

  Perform general communication initialization from Constructor.
- virtual ~**Worker** ()

  ***Worker** (p. 828) destructor.*

Static Public Attributes

- static const std::string **PARAM_PORT**

Additional Inherited Members

Protected Member Functions inherited from BiometricEvaluation::Process::Worker

- **Worker** ()

  Worker *(p. 828) constructor.*
- virtual bool **stopRequested** () const final

  Determine if the parent has requested this child to exit.
- bool **waitForMessage** (int numSeconds=-1) const

  Block while waiting for a message from the ***Manager*** *(p. 596).*

H.88.1 Detailed Description

Accepts new connections and spawns message receivers.

H.88.2 Member Function Documentation

H.88.2.1 workerMain()

`int32_t BiometricEvaluation::Process::MessageCenterListener::workerMain () [virtual]`

The method that will get called to start execution by a ProcessManager.

Returns

Status code.

Note

If an object of this class is added to a **Process::ForkManager** (p. 447) object, the implementation of **Process::Worker::workerMain()** (p. 834) should release all resources prior to returning.

Any exceptions thrown by this method will cause the worker to exit with a return status of EXIT_↩ FAILURE. The type and contents of the exception is not maintained.

Implements **BiometricEvaluation::Process::Worker** (p. 834).

H.88.3 Member Data Documentation

H.88.3.1 PARAM_PORT

`const std::string BiometricEvaluation::Process::MessageCenterListener::PARAM_PORT [static]`

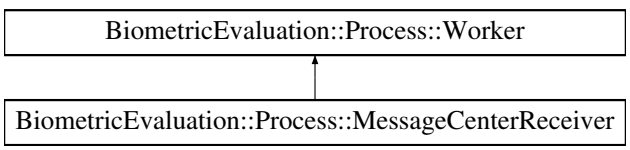
Parameter used to pass port number

H.89 BiometricEvaluation::Process::MessageCenterReceiver Class Reference

Receives message from a client, forwarding to the central **MessageCenter** (p. 607).

`#include <be_process_mcreceiver.h>`

Inheritance diagram for BiometricEvaluation::Process::MessageCenterReceiver:



Public Member Functions

- int32_t **workerMain** ()
- **MessageCenterReceiver** ()=default
- ~**MessageCenterReceiver** ()=default

Public Member Functions inherited from BiometricEvaluation::Process::Worker

- std::shared_ptr< void > **getParameter** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828).*
- double **getParameterAsDouble** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828) as a double.*
- int64_t **getParameterAsInteger** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828) as an integer.*
- std::string **getParameterAsString** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828) as a string.*
- void **setParameter** (const std::string &name, std::shared_ptr< void > argument)

  *Pass a parameter to this **Worker** (p. 828).*
- virtual void **stop** () final

  *Tell this **Worker** (p. 828) to return ASAP.*
- void **closeWorkerPipeEnds** ()

  *Perform initialization for communication from **Worker** (p. 828) to **Manager** (p. 596).*
- void **closeManagerPipeEnds** ()

  *Perform initialization for communication from **Manager** (p. 596) to **Worker** (p. 828).*
- int **getSendingPipe** () const

  *Obtain the pipe used to send messages to this **Worker** (p. 828).*
- int **getReceivingPipe** () const

  *Obtain the pipe used to receive messages to this **Worker** (p. 828).*
- void **sendMessageToManager** (const **Memory::uint8Array** &message)

  *Send a message to the **Manager** (p. 596).*
- void **receiveMessageFromManager** (**Memory::uint8Array** &message)

  *Receive a message from the **Manager** (p. 596).*
- void **_initCommunication** ()

  Perform general communication initialization from Constructor.
- virtual ~**Worker** ()

  ***Worker** (p. 828) destructor.*

Static Public Attributes

- static const std::string **PARAM_CLIENT_SOCKET**
- static const std::string **PARAM_CLIENT_ID**
- static const std::string **MSG_DISCONNECT**

Additional Inherited Members

Protected Member Functions inherited from BiometricEvaluation::Process::Worker

- **Worker** ()

  Worker *(*p. *828) constructor.*
- virtual bool **stopRequested** () const final

  Determine if the parent has requested this child to exit.
- bool **waitForMessage** (int numSeconds=-1) const

  Block while waiting for a message from the ***Manager*** *(*p. *596).*

H.89.1 Detailed Description

Receives message from a client, forwarding to the central **MessageCenter** (p. 607).

H.89.2 Constructor & Destructor Documentation

H.89.2.1 MessageCenterReceiver()

`BiometricEvaluation::Process::MessageCenterReceiver::MessageCenterReceiver () [default]`

Default constructor.

H.89.2.2 ~MessageCenterReceiver()

`BiometricEvaluation::Process::MessageCenterReceiver::~MessageCenterReceiver () [default]`

Default destructor.

H.89.3 Member Function Documentation

H.89.3.1 workerMain()

`int32_t BiometricEvaluation::Process::MessageCenterReceiver::workerMain () [virtual]`

Receive loop.

Implements **BiometricEvaluation::Process::Worker** (p. 834).

H.89.4 Member Data Documentation

H.89.4.1 MSG_DISCONNECT

`const std::string BiometricEvaluation::Process::MessageCenterReceiver::MSG_DISCONNECT [static]`

Message sent when client should disconnect.

H.89.4.2 PARAM_CLIENT_ID

`const std::string BiometricEvaluation::Process::MessageCenterReceiver::PARAM_CLIENT_ID [static]`

Parameter used to pass an ID to the client.

H.89.4.3 PARAM_CLIENT_SOCKET

`const std::string BiometricEvaluation::Process::MessageCenterReceiver::PARAM_CLIENT_SOCKET [static]`

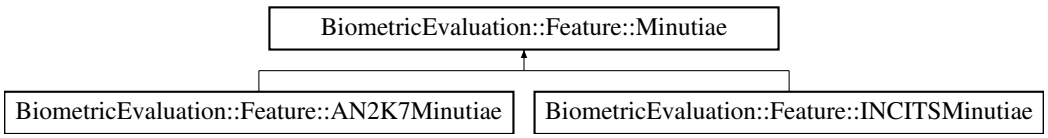
Parameter used to pass client socket FD.

H.90 BiometricEvaluation::Feature::Minutiae Class Reference

A class to represent a set of minutiae data points.

`#include <be_feature_minutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::Minutiae:



Public Member Functions

- virtual **MinutiaeFormat** **getFormat** () const =0

  Obtain the minutiae format kind.
- virtual MinutiaPointSet **getMinutiaPoints** () const =0

  Obtain the set of finger minutiae data points. The set may be empty.
- virtual RidgeCountItemSet **getRidgeCountItems** () const =0

  Obtain the set of ridge count data items. The set may be empty.
- virtual CorePointSet **getCores** () const =0

  Obtains the set of core positions. The set may be empty.
- virtual DeltaPointSet **getDeltas** () const =0

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

H.90.1 Detailed Description

A class to represent a set of minutiae data points.

Each set includes the core and delta data points, if they are included in the source record. This class represents an interface that subclasses of this class will implement, providing more information on the minutioe that is specific to the record format represented by that class.

H.90.2 Member Function Documentation

H.90.2.1 getCores()

```
virtual CorePointSet BiometricEvaluation::Feature::Minutiae::getCores () const  [pure virtual]
```

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

Implemented in **BiometricEvaluation::Feature::AN2K7Minutiae** (p. 198), and **BiometricEvaluation↩ ::Feature::INCITSMinutiae** (p. 497).

H.90.2.2 getDeltas()

```
virtual DeltaPointSet BiometricEvaluation::Feature::Minutiae::getDeltas () const  [pure virtual]
```

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

Implemented in **BiometricEvaluation::Feature::AN2K7Minutiae** (p. 198), and **BiometricEvaluation↩ ::Feature::INCITSMinutiae** (p. 497).

H.90.2.3 getFormat()

```
virtual MinutiaeFormat BiometricEvaluation::Feature::Minutiae::getFormat () const [pure virtual]
```

Obtain the minutiae format kind.

Implemented in **BiometricEvaluation::Feature::AN2K7Minutiae** (p. 198), and **BiometricEvaluation::Feature::INCITSMinutiae** (p. 497).

H.90.2.4 getMinutiaPoints()

```
virtual MinutiaPointSet BiometricEvaluation::Feature::Minutiae::getMinutiaPoints () const [pure virtual]
```

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

Implemented in **BiometricEvaluation::Feature::AN2K7Minutiae** (p. 198), and **BiometricEvaluation::Feature::INCITSMinutiae** (p. 498).

H.90.2.5 getRidgeCountItems()

```
virtual RidgeCountItemSet BiometricEvaluation::Feature::Minutiae::getRidgeCountItems () const [pure virtual]
```

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

Implemented in **BiometricEvaluation::Feature::AN2K7Minutiae** (p. 199), and **BiometricEvaluation::Feature::INCITSMinutiae** (p. 498).

H.91 BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount Struct Reference

Representation of an extended feature set ridge count info.

```
#include <be_feature_an2k11efs.h>
```

Public Attributes

- int **mia**
- int **mib**
- int **mir**
- bool **has_mrn**
- int **mrn**
- bool **has_mrs**
- int **mrs**

H.91.1 Detailed Description

Representation of an extended feature set ridge count info.

H.91.2 Member Data Documentation

H.91.2.1 mia

```
int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mia
```

minutia index A

H.91.2.2 mib

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mib`
minutia index B

H.91.2.3 mir

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mir`
ridge count

H.91.2.4 mrn

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mrn`
reference number, optional

H.91.2.5 mrs

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mrs`
residual, optional

H.92 BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCountConfidence Struct Reference

Representation of an extended feature set minutiae ridge count confidence item.
`#include <be_feature_an2k11efs.h>`

Public Attributes

- **Image::Coordinate pointA**
- **Image::Coordinate pointB**
- MethodOfRidgeCounting **morc**
- int **mcv**

H.92.1 Detailed Description

Representation of an extended feature set minutiae ridge count confidence item.

H.93 BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCountInfo Struct Reference

All the ridge count information in one place.
`#include <be_feature_an2k11efs.h>`

Public Attributes

- bool **has_mra**
- MinutiaeRidgeCountAlgorithm **mra**
- bool **has_mrcs**
- MinutiaeRidgeCountSet **mrcs**
- bool **has_rccs**
- MinutiaeRidgeCountConfidenceSet **rccs**

H.93.1 Detailed Description

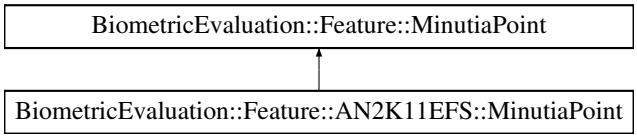
All the ridge count information in one place.

H.94 BiometricEvaluation::Feature::AN2K11EFS::MinutiaPoint Struct Reference

Representation of an extended feature set minutia data point.

`#include <be_feature_an2k11efs.h>`

Inheritance diagram for BiometricEvaluation::Feature::AN2K11EFS::MinutiaPoint:



Public Attributes

- bool **has_mru**
- int **mru**
- bool **has_mdu**
- int **mdu**

Public Attributes inherited from BiometricEvaluation::Feature::MinutiaPoint

- unsigned int **index**
- bool **has_type**
- **MinutiaeType type**
- **Image::Coordinate coordinate**
- unsigned int **theta**
- bool **has_quality**
- unsigned int **quality**

H.94.1 Detailed Description

Representation of an extended feature set minutia data point.

H.94.2 Member Data Documentation

H.94.2.1 mdu

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaPoint::mdu`

minutiae direction uncertainty

H.94.2.2 mru

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaPoint::mru`

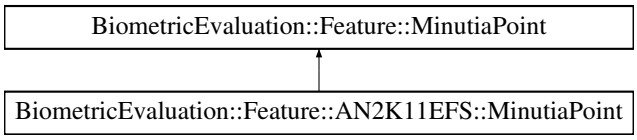
radius of position uncertainty

H.95 BiometricEvaluation::Feature::MinutiaPoint Struct Reference

Representation of a finger minutiae data point.

`#include <be_feature_minutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::MinutiaPoint:



Public Attributes

- unsigned int **index**
- bool **has_type**
- **MinutiaeType type**
- **Image::Coordinate coordinate**
- unsigned int **theta**
- bool **has_quality**
- unsigned int **quality**

H.95.1 Detailed Description

Representation of a finger minutiae data point.

H.96 BiometricEvaluation::Feature::MPEGFacePoint Struct Reference

Representation of a feature point and a set of points.

`#include <be_feature_mpegfacepoint.h>`

Public Attributes

- uint8_t **type**
- uint8_t **major**
- uint8_t **minor**
- **BiometricEvaluation::Image::Coordinate coordinate**

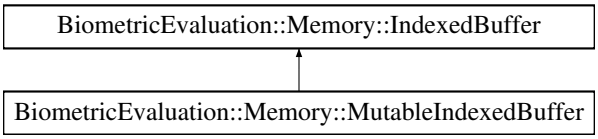
H.96.1 Detailed Description

Representation of a feature point and a set of points.

H.97 BiometricEvaluation::Memory::MutableIndexedBuffer Class Reference

`#include <be_memory_mutableindexedbuffer.h>`

Inheritance diagram for BiometricEvaluation::Memory::MutableIndexedBuffer:



Public Member Functions

- **MutableIndexedBuffer** (uint8_t *data, uint64_t size)

  Wrap an existing buffer of a given length.
- **MutableIndexedBuffer** (**uint8Array** &aa)

  Wrap an existing uint8Array.
- **MutableIndexedBuffer** (const **MutableIndexedBuffer** &copy)=default
- uint64_t **push** (const void *buf, uint64_t len)

  Push elements into the buffer, inreasing the index.
- uint8_t **pushU8Val** (uint8_t val)

  Push an element into the managed buffer at the current index, incrementing the index.
- uint16_t **pushU16Val** (uint16_t val)

  Push two elements into the managed buffer at the current index, incrementing the index.
- uint16_t **pushBeU16Val** (uint16_t val)

  Push two elements into the managed buffer at the current index as a big endian value, incrementing the index.
- uint32_t **pushU32Val** (uint32_t val)

  Push four elements into the managed buffer at the current index, incrementing the index.
- uint32_t **pushBeU32Val** (uint32_t val)

  Push four elements into the managed buffer at the current index as a big endian value, incrementing the index.
- uint64_t **pushU64Val** (uint64_t val)

  Push eight elements into the managed buffer at the current index, incrementing the index.
- virtual const uint8_t * **get** () const

  Returns a pointer to the managed buffer.
- virtual ~**MutableIndexedBuffer** ()=default

Public Member Functions inherited from BiometricEvaluation::Memory::IndexedBuffer

- **IndexedBuffer** ()
- **IndexedBuffer** (const uint8_t *data, uint64_t size)

  Wrap an existing buffer of a given length.
- **IndexedBuffer** (const **uint8Array** &aa)

  Wrap an existing uint8Array.
- **IndexedBuffer** (const **IndexedBuffer** &copy)=default
- uint32_t **getSize** () const

  Obtain the current size of the buffer.
- uint32_t **getIndex** () const

  Obtain the current index into the buffer.
- void **setIndex** (uint64_t index)

  Set the current index into the buffer.

- uint8_t **scanU8Val** ()

  Obtain the next element of the buffer and increment the current index value.
- uint16_t **scanU16Val** ()

  Obtain the next two elements of the buffer and increment the current index value.
- uint16_t **scanBeU16Val** ()

  Obtain the next two elements of the buffer, scanned as a big-endian value, and increment the current index value.
- uint32_t **scanU32Val** ()

  Obtain the next four elements of the buffer and increment the current index value by four.
- uint32_t **scanBeU32Val** ()

  Obtain the next four elements of the buffer, scanned as a big-endian value, and increment the current index value.
- uint64_t **scanU64Val** ()

  Obtain the next eight elements of the buffer and increment the current index value by eight.
- uint64_t **scan** (void *buf, uint64_t len)

  Obtain the next 'n' elements of the buffer and increment the current index value by n.
- virtual ~**IndexedBuffer** ()=default

H.97.1 Detailed Description

Mutable version of an **IndexedBuffer** (p. 539).

H.97.2 Constructor & Destructor Documentation

H.97.2.1 MutableIndexedBuffer() [1/3]

```
BiometricEvaluation::Memory::MutableIndexedBuffer::MutableIndexedBuffer (
            uint8_t * data,
            uint64_t size)
```

Wrap an existing buffer of a given length.

Parameters

data	Buffer to wrap.
size	Size of buffer.

H.97.2.2 MutableIndexedBuffer() [2/3]

```
BiometricEvaluation::Memory::MutableIndexedBuffer::MutableIndexedBuffer (
             uint8Array & aa)
```

Wrap an existing uint8Array.

Parameters

aa	uint8↩ Array to wrap.

H.97.2.3 MutableIndexedBuffer() [3/3]

```
BiometricEvaluation::Memory::MutableIndexedBuffer::MutableIndexedBuffer (
            const MutableIndexedBuffer & copy) [default]
```

Copy constructor (default).

H.97.2.4 ~MutableIndexedBuffer()

```
virtual BiometricEvaluation::Memory::MutableIndexedBuffer::~MutableIndexedBuffer () [virtual],
[default]
```

Destructor (default).

H.97.3 Member Function Documentation

H.97.3.1 get()

```
virtual const uint8_t * BiometricEvaluation::Memory::MutableIndexedBuffer::get () const [virtual]
```

Returns a pointer to the managed buffer.

Returns

Pointer to the managed buffer.

Reimplemented from **BiometricEvaluation::Memory::IndexedBuffer** (p. 541).

H.97.3.2 push()

```
uint64_t BiometricEvaluation::Memory::MutableIndexedBuffer::push (
            const void * buf,
            uint64_t len)
```

Push elements into the buffer, inreasing the index.

Parameters

in	*buf*	The buffer to push. If nullptr, 0 will be inserted.
in	*len*	The number of elements from buf to copy.

Exceptions

Error::DataError *(p. 390)*	Not enough room to copy len elements.

Returns

The number of elements copied.

H.97.3.3 pushBeU16Val()

```
uint16_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushBeU16Val (
            uint16_t val)
```

Push two elements into the managed buffer at the current index as a big endian value, incrementing the index.

Parameters

val	Value to push.

Exceptions

Error::DataError *(p. 390)*	Not enough room to copy the elements.

Returns

The number of elements copied (2).

H.97.3.4 pushBeU32Val()

```
uint32_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushBeU32Val (
            uint32_t val)
```

Push four elements into the managed buffer at the current index as a big endian value, incrementing the index.

Parameters

val	Value to push.

Exceptions

Error::DataError *(p. 390)*	Not enough room to copy the elements.

Returns

The number of elements copied (4).

H.97.3.5 pushU16Val()

```
uint16_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushU16Val (
            uint16_t val)
```

Push two elements into the managed buffer at the current index, incrementing the index.

Parameters

val	Value to push.

Exceptions

Error::DataError *(p. 390)*	Not enough room to copy the elements.

Returns

The number of elements copied (2).

H.97.3.6 pushU32Val()

```
uint32_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushU32Val (
            uint32_t val)
```

Push four elements into the managed buffer at the current index, incrementing the index.

Parameters

val	Value to push.

Exceptions

Error::DataError *(p. 390)*	Not enough room to copy the elements.

Returns

The number of elements copied (4).

H.97.3.7 pushU64Val()

```
uint64_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushU64Val (
            uint64_t val)
```

Push eight elements into the managed buffer at the current index, incrementing the index.

Parameters

val	Value to push.

Exceptions

Error::DataError *(p. 390)*	Not enough room to copy the elements.

Returns

The number of elements copied (8).

H.97.3.8 pushU8Val()

```
uint8_t BiometricEvaluation::Memory::MutableIndexedBuffer::pushU8Val (
            uint8_t val)
```

Push an element into the managed buffer at the current index, incrementing the index.

Parameters

val	Value to push.

Exceptions

Error::DataError *(p. 390)*	Not enough room to copy the element.

Returns

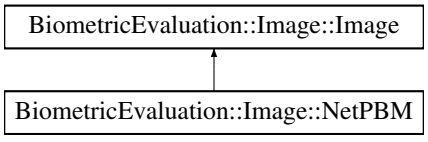
The number of elements copied (1).

H.98 BiometricEvaluation::Image::NetPBM Class Reference

A NetPBM-encoded image.

`#include <be_image_netpbm.h>`

Inheritance diagram for BiometricEvaluation::Image::NetPBM:



Public Types

- enum class **Kind** {
  ASCIIPortableBitmap = 1 , **ASCIIPortableGraymap** = 2 , **ASCIIPortablePixmap** = 3 , **Binary↩ PortableBitmap** = 4 ,
  BinaryPortableGraymap = 5 , **BinaryPortablePixmap** = 6 }

Public Types inherited from BiometricEvaluation::Image::Image

- using **statusCallback_t**

Public Member Functions

- **NetPBM** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback↩ _t &statusCallback= **Image::defaultStatusCallback**)
- **NetPBM** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Memory::uint8Array** **getRawData** () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::uint8Array** **getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.

Public Member Functions inherited from BiometricEvaluation::Image::Image

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image↩ ::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm** **getCompressionAlgorithm** () const

  Accessor for the CompressionAlgorithm of the image.
- **Resolution** **getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array** **getData** () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual **Memory::uint8Array** **getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Size** **getDimensions** () const

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

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

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

  Accessor for the presence of an alpha channel.
- statusCallback_t **getStatusCallback** () const

Get handle to status callback function.

- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Static Public Member Functions

- static bool **isNetPBM** (const uint8_t *data, uint64_t size)
- static void **skipLine** (const uint8_t *data, size_t dataSize, size_t &offset)

  Skip an entire line of input, placing offset at the first character after the newline.
- static void **skipComment** (const uint8_t *data, size_t dataSize, size_t &offset)

  Skip a block of comments in input.
- static std::string **getNextValue** (const uint8_t *data, size_t dataSize, size_t &offset, size_t sizeOfValue=0)

  Obtain the next space-separated value from data, beginning at offset.
- static **Memory::uint8Array** **ASCIIBitmapTo8Bit** (const uint8_t *bitmap, uint64_t bitmapSize, uint32_t width, uint32_t height)

  Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.
- static **Memory::uint8Array** **ASCIIPixmapToBinaryPixmap** (const uint8_t *ASCIIBuf, uint64_t ASCIIBufSize, uint32_t width, uint32_t height, uint8_t depth, uint32_t maxColor)

  Convert an ASCII pixel map buffer into a binary pixel map buffer.
- static **Memory::uint8Array** **BinaryBitmapTo8Bit** (const uint8_t *bitmap, uint64_t bitmapSize, uint32_t width, uint32_t height)

  Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Static Public Member Functions inherited from BiometricEvaluation::Image::Image

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.
- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of an image file and create an **Image** (p. 477) object.*
- static **CompressionAlgorithm** **getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm** **getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm** **getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.
- static **BiometricEvaluation::Image::Raw** **getRawImage** (const std::shared_ptr< **BiometricEvaluation::Image::Image** > &image)

  *Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).*
- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Additional Inherited Members

Protected Member Functions inherited from BiometricEvaluation::Image::Image

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .
- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.
- void **setColorDepth** (const uint32_t colorDepth)

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

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

  Mutator for the presence of an alpha channel.

H.98.1 Detailed Description

A NetPBM-encoded image.

Note

While a **NetPBM** (p. 625) 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** (p. 625) format.

H.98.2 Member Function Documentation

H.98.2.1 ASCIIBitmapTo8Bit()

```
static Memory::uint8Array BiometricEvaluation::Image::NetPBM::ASCIIBitmapTo8Bit (
            const uint8_t * bitmap,
            uint64_t bitmapSize,
            uint32_t width,
            uint32_t height)  [static]
```

Convert an ASCII bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Parameters

bitmap	Bitmap data buffer.
bitmapSize	**Size** (p. 763) of bitmap.
width	Width of image in bitmap.

Parameters

height	Height of image in bitmap.

Returns

8-bit depth representation of bitmap

Exceptions

out_of_range	**Error** (p. 112) extracting a value from the bitmap.

H.98.2.2 ASCIIPixmapToBinaryPixmap()

```
static Memory::uint8Array BiometricEvaluation::Image::NetPBM::ASCIIPixmapToBinaryPixmap (
            const uint8_t * ASCIIBuf,
            uint64_t ASCIIBufSize,
            uint32_t width,
            uint32_t height,
            uint8_t depth,
            uint32_t maxColor) [static]
```

Convert an ASCII pixel map buffer into a binary pixel map buffer.

Parameters

ASCIIBuf	ASCII pixel map data buffer.
ASCIIBufSize	**Size** (p. 763) of ASCIIBuf.
width	Width of image in pixel map.
height	Height of image in pixel map.

Parameters

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** (p. 112) extracting a value from the pixel map.
Error::ParameterError *(p. 655)*	Invalid value for depth, must be a multiple of 8.

H.98.2.3 BinaryBitmapTo8Bit()

```
static Memory::uint8Array BiometricEvaluation::Image::NetPBM::BinaryBitmapTo8Bit (
            const uint8_t * bitmap,
            uint64_t bitmapSize,
            uint32_t width,
            uint32_t height) [static]
```

Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Parameters

bitmap	Bitmap data buffer.
bitmapSize	**Size** (p. 763) of bitmap.

Parameters

width	Width of image in bitmap.
height	Height of image in bitmap.

Returns

8-bit depth representation of bitmap

Exceptions

out_of_range	**Error** (p. 112) extracting a value from the bitmap.

H.98.2.4 getNextValue()

```
static std::string BiometricEvaluation::Image::NetPBM::getNextValue (
            const uint8_t * data,
            size_t dataSize,
            size_t & offset,
            size_t sizeOfValue = 0)  [static]
```

Obtain the next space-separated value from data, beginning at offset.

Parameters

data	Buffer where next value will be obtained.
dataSize	**Size** (p. 763) of data.
offset	Current starting position within data.

Parameters

sizeOfValue	In the event that the values in data are not space-separated, return a value when it reaches sizeOf Value length. 0 assumes space-separated.

Returns

Next value from data.

H.98.2.5 getRawData()

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

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

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.
Error::NotImplemented (p. *636)*	Compression type not supported.

Note

The raw data returned from this method is encoded at the same bit depth as the compressed data, except in the case of 1-bit (bitmap) images, which are expanded to 8-bit.

Implements **BiometricEvaluation::Image::Image** (p. 486).

H.98.2.6 getRawGrayscaleData()

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

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.
Error::NotImplemented *(p. 636)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 655)*	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implements **BiometricEvaluation::Image::Image** (p. 487).

H.98.2.7 isNetPBM()

```
static bool BiometricEvaluation::Image::NetPBM::isNetPBM (
            const uint8_t * data,
            uint64_t size)  [static]
```

Whether or not data is a netpbm image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a netpbm image, false otherwise.

H.98.2.8 skipComment()

```
static void BiometricEvaluation::Image::NetPBM::skipComment (
            const uint8_t * data,
            size_t dataSize,
            size_t & offset)  [static]
```

Skip a block of comments in input.

Parameters

data	Buffer with comment to be skipped.
dataSize	**Size** (p. 763) of data
offset	Position within data from which the rest of the line should be read.

Exceptions

out_of_range	End of line not encountered before end of data or on last line of data.

H.98.2.9 skipLine()

```
static void BiometricEvaluation::Image::NetPBM::skipLine (
            const uint8_t * data,
            size_t dataSize,
            size_t & offset)  [static]
```

Skip an entire line of input, placing offset at the first character after the newline.

Parameters

data	Buffer with line to be skipped.
dataSize	**Size** (p. 763) of data.
offset	Position within data from which the rest of the line should be read.

Exceptions

out_of_range	End of line not encountered before end of data or on last line of data.

H.99 BiometricEvaluation::Feature::AN2K11EFS::NoFeatures↩ Present Struct Reference

A set of flags indicating "No features present" indicators contained within the extended feature set.

```
#include <be_feature_an2k11efs.h>
```

Public Attributes

- bool **cores**
- bool **deltas**
- bool **minutiae**

H.99.1 Detailed Description

A set of flags indicating "No features present" indicators contained within the extended feature set.

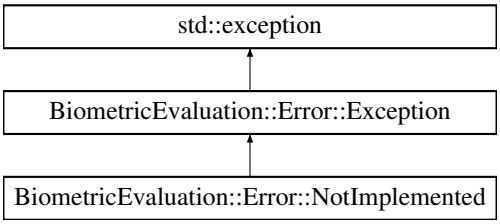
A flag is set to true when the Type-9 field is set to 'Y', indicating that analysis of the image has determined that there are no instances of that feature present in the image. Otherwise the Type-9 field is is not present and the flag will be false.

H.100 BiometricEvaluation::Error::NotImplemented Class Reference

A **NotImplemented** (p. 636) 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.100.1 Detailed Description

A **NotImplemented** (p. 636) object is thrown when the underlying implementation of this interface has not or could not be created.

H.100.2 Constructor & Destructor Documentation

H.100.2.1 NotImplemented() [1/2]

```
BiometricEvaluation::Error::NotImplemented::NotImplemented ()
```

Construct a **NotImplemented** (p. 636) object with the default information string.

H.100.2.2 NotImplemented() [2/2]

```
BiometricEvaluation::Error::NotImplemented::NotImplemented (
            const std::string & info)
```

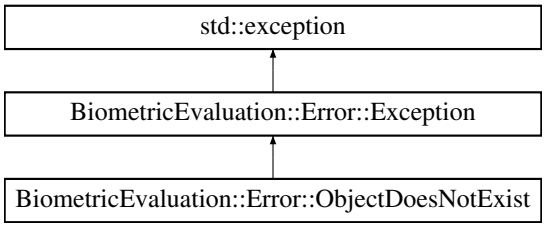
Construct a **NotImplemented** (p. 636) object with an information string appended to the default information string.

H.101 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.101.1 Detailed Description

The named object does not exist.

H.101.2 Constructor & Destructor Documentation

H.101.2.1 ObjectDoesNotExist() [1/2]

```
BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ()
```

Construct a **ObjectDoesNotExist** (p. 637) object with the default information string.

H.101.2.2 ObjectDoesNotExist() [2/2]

```
BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist (
            const std::string & info)
```

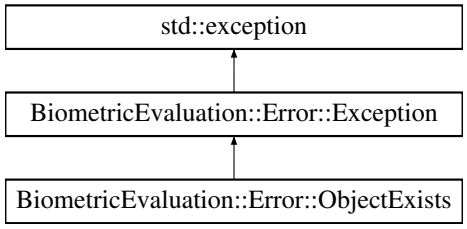
Construct a **ObjectDoesNotExist** (p. 637) object with an information string appended to the default information string.

H.102 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.102.1 Detailed Description

The named object exists and will not be replaced.

H.102.2 Constructor & Destructor Documentation

H.102.2.1 ObjectExists() [1/2]

```
BiometricEvaluation::Error::ObjectExists::ObjectExists ()
```

Construct a **ObjectExists** (p. 637) object with the default information string.

H.102.2.2 ObjectExists() [2/2]

```
BiometricEvaluation::Error::ObjectExists::ObjectExists (
            const std::string & info)
```

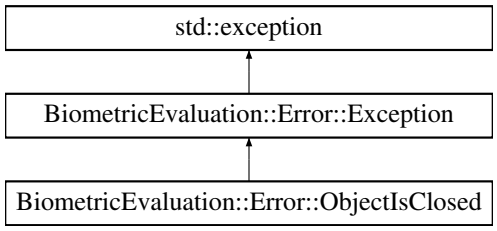
Construct a **ObjectExists** (p. 637) object with an information string appended to the default information string.

H.103 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.103.1 Detailed Description

The object is closed.

H.103.2 Constructor & Destructor Documentation

H.103.2.1 ObjectIsClosed() [1/2]

```
BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ()
```

Construct a **ObjectIsClosed** (p. 638) object with the default information string.

H.103.2.2 ObjectIsClosed() [2/2]

```
BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed (
            const std::string & info)
```

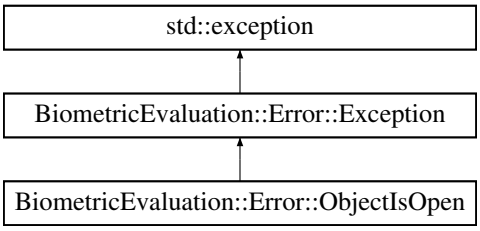
Construct a **ObjectIsClosed** (p. 638) object with an information string appended to the default information string.

H.104 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.104.1 Detailed Description

The object is already opened.

H.104.2 Constructor & Destructor Documentation

H.104.2.1 ObjectIsOpen() [1/2]

```
BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ()
```

Construct a **ObjectIsOpen** (p. 639) object with the default information string.

H.104.2.2 ObjectIsOpen() [2/2]

```
BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen (
            const std::string & info)
```

Construct a **ObjectIsOpen** (p. 639) object with an information string appended to the default information string.

H.105 BiometricEvaluation::Memory::OrderedMap< Key, T > Class Template Reference

```
#include <be_memory_orderedmap.h>
```

Public Types

- using **container** = typename std::unordered_map<Key, T>
- using **iterator** = **OrderedMapIterator**<Key, T>
- using **const_iterator** = **OrderedMapConstIterator**<Key, T>
- using **size_type** = typename container::size_type
- using **value_type** = typename container::value_type

- using **key_type** = Key
- using **mapped_type** = T
- using **key_equal** = typename container::key_equal

Public Member Functions

- **OrderedMap** ()
- bool **push_back** (const value_type &value)

  Insert an element at the end of the collection.
- void **erase** (**iterator** pos)

  Remove an element from the collection.
- void **erase** (const Key &key)

  Remove an element from the collection.
- **iterator begin** ()
- **const_iterator begin** () const
- **const_iterator cbegin** () const
- **iterator end** ()
- **const_iterator end** () const
- **const_iterator cend** () const
- size_type **size** () const
- bool **keyExists** (const Key &key) const

  Determine if a value exists in the container.
- const **OrderedMapIterator**< Key, T > **find** (const Key &key) const

  Obtain an iterator to a particular key.
- std::shared_ptr< value_type > **find_quick** (const Key &key) const
- T & **operator[ ]** (const Key &key)

  Subscripting operator.
- key_equal **key_eq** () const
- ~**OrderedMap** ()

Friends

- class **OrderedMapIterator**< **Key, T** >
- class **OrderedMapConstIterator**< **Key, T** >

H.105.1 Detailed Description

template<class Key, class T>
class BiometricEvaluation::Memory::OrderedMap< Key, T >

A map where insertion order is preserved and elements are unique.

H.105.2 Constructor & Destructor Documentation

H.105.2.1 OrderedMap()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::OrderedMap ()
```

Constructor.

H.105.2.2 ~OrderedMap()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::~ OrderedMap ()
```

Destructor

H.105.3 Member Function Documentation

H.105.3.1 begin() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::iterator BiometricEvaluation::Memory::↩
OrderedMap< Key, T >::begin ()
```

Returns

Iterator at the first element of the collection.

H.105.3.2 begin() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory↩
::OrderedMap< Key, T >::begin () const
```

Returns

Iterator at the first element of the collection.

H.105.3.3 cbegin()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory↩
::OrderedMap< Key, T >::cbegin () const
```

Returns

Iterator at the first element of the collection.

H.105.3.4 cend()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory↩
::OrderedMap< Key, T >::cend () const
```

Returns

Iterator beyond the last element of the collection.

H.105.3.5 end() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::iterator BiometricEvaluation::Memory::↩
OrderedMap< Key, T >::end ()
```

Returns

Iterator beyond the last element of the collection.

H.105.3.6 end() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::const_iterator BiometricEvaluation::Memory↩
::OrderedMap< Key, T >::end () const
```

Returns

Iterator beyond the last element of the collection.

H.105.3.7 erase() [1/2]

```
template<class Key , class T >
void BiometricEvaluation::Memory::OrderedMap< Key, T >::erase (
            const Key & key)
```

Remove an element from the collection.

Parameters

key	Key of the element to remove.

H.105.3.8 erase() [2/2]

```
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()).

H.105.3.9 find()

```
template<class Key , class T >
const  BiometricEvaluation::Memory::OrderedMapIterator< Key, T >  BiometricEvaluation::Memory↩
::OrderedMap< Key, T >::find (
            const Key & key) const
```

Obtain an iterator to a particular key.

Note

Complexity is O(n).

H.105.3.10 key_eq()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::key_equal  BiometricEvaluation::Memory::↩
OrderedMap< Key, T >::key_eq () const
```

Returns

Function that compares keys for equality.

H.105.3.11 keyExists()

```
template<class Key , class T >
bool  BiometricEvaluation::Memory::OrderedMap< Key, T >::keyExists (
            const Key & key) const
```

Determine if a value exists in the container.

Parameters

key	Key to search the container for.

Returns

Whether or not key exists in this container.

Note

Complexity is O(1).

H.105.3.12 operator[]()

```
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.

H.105.3.13 push_back()

```
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()).

H.105.3.14 size()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::size_type  BiometricEvaluation::Memory::↩
OrderedMap< Key, T >::size () const
```

Returns

Number of elements in the collection.

H.106 BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > Class Template Reference

```
#include <be_memory_orderedmap.h>
```

Public Types

- using **iterator_category**
- using **value_type** = std::pair<Key, T>
- using **difference_type** = std::ptrdiff_t
- using **pointer** = const **value_type***
- using **reference** = const **value_type**&

Public Member Functions

- **OrderedMapConstIterator** ()
- **OrderedMapConstIterator** (const **OrderedMapIterator**< Key, T > &iterator)
- ~**OrderedMapConstIterator** ()
- **reference operator*** () const
- **pointer operator->** () const
- **OrderedMapConstIterator** & **operator++** ()
- **OrderedMapConstIterator operator++** (int)
- **OrderedMapConstIterator** & **operator--** ()
- **OrderedMapConstIterator operator--** (int)
- bool **operator==** (const **OrderedMapConstIterator** &rhs) const

  Test for iterator equality.
- bool **operator!=** (const **OrderedMapConstIterator** &rhs) const

  Test for iterator equality.

Friends

- class **OrderedMap**< **Key, T** >

H.106.1 Detailed Description

template<class Key, class T>
class BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >

Const Iterator for OrderedMaps.

H.106.2 Member Typedef Documentation

H.106.2.1 difference_type

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::difference_type = std↩
::ptrdiff_t
```

Type used to measure distance between iterators

H.106.2.2 iterator_category

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::iterator_category
```

Initial value:

```
std::bidirectional_iterator_tag
```

Type of iterator

H.106.2.3 pointer

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::pointer = const value_type*
```

Pointer to the type iterated over

H.106.2.4 reference

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::reference = const value_type&
```

Reference to the type iterated over

H.106.2.5 value_type

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::value_type = std::pair<Key, T>
```

Type when dereferencing iterators

H.106.3 Constructor & Destructor Documentation

H.106.3.1 OrderedMapConstIterator() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::OrderedMapConstIterator ()
```

Constructor

H.106.3.2 OrderedMapConstIterator() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::OrderedMapConstIterator (
            const OrderedMapIterator< Key, T > & iterator)
```

Iterator to ConstIterator converter

H.106.3.3 ~OrderedMapConstIterator()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::~ OrderedMapConstIterator ()
```

Destructor

H.106.4 Member Function Documentation

H.106.4.1 operator"!=()

```
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.

H.106.4.2 operator*()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > ::reference  BiometricEvaluation↩
::Memory::OrderedMapConstIterator< Key, T >::operator* () const
```

Returns

Reference to the current iterated pair.

H.106.4.3 operator++() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > &  BiometricEvaluation::Memory↩
::OrderedMapConstIterator< Key, T >::operator++ ()
```

Move to the next pair

H.106.4.4 operator++() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >  BiometricEvaluation::Memory↩
::OrderedMapConstIterator< Key, T >::operator++ (
            int )
```

Move to the next pair

H.106.4.5 operator--() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > &  BiometricEvaluation::Memory↩
::OrderedMapConstIterator< Key, T >::operator-- ()
```

Move to the previous pair.

H.106.4.6 operator--() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >  BiometricEvaluation::Memory↩
::OrderedMapConstIterator< Key, T >::operator-- (
            int )
```

Move to the previous pair.

H.106.4.7 operator->()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > ::pointer BiometricEvaluation↩
::Memory::OrderedMapConstIterator< Key, T >::operator-> () const
```

Returns

Pointer to the current iterated pair.

H.106.4.8 operator==()

```
template<class Key , class T >
bool BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::operator== (
        const OrderedMapConstIterator< Key, T > & rhs) const
```

Test for iterator equality.

Parameters

rhs	Object on the right-hand side of the expression.

Returns

Whether or not this iterator is equivalent to rhs.

H.107 BiometricEvaluation::Memory::OrderedMapIterator< Key, T > Class Template Reference

```
#include <be_memory_orderedmap.h>
```

Public Types

- using **iterator_category**
- using **value_type** = std::pair<Key, T>
- using **difference_type** = std::ptrdiff_t
- using **pointer** = **value_type***
- using **reference** = **value_type**&

Public Member Functions

- **OrderedMapIterator** ()
- ~**OrderedMapIterator** ()
- **reference operator*** () const
- **pointer operator->** () const

- **OrderedMapIterator** & **operator++** ()
- **OrderedMapIterator** **operator++** (int)
- **OrderedMapIterator** & **operator--** ()
- **OrderedMapIterator** **operator--** (int)
- bool **operator==** (const **OrderedMapIterator** &rhs) const

  Test for iterator equality.
- bool **operator!=** (const **OrderedMapIterator** &rhs) const

  Test for iterator equality.

Friends

- class **OrderedMap**< **Key, T** >
- class **OrderedMapConstIterator**< **Key, T** >

H.107.1 Detailed Description

template<class Key, class T>
class BiometricEvaluation::Memory::OrderedMapIterator< Key, T >

Iterator for OrderedMaps.

H.107.2 Member Typedef Documentation

H.107.2.1 difference_type

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::difference_type = std::ptrdiff↩
_t
```

Type used to measure distance between iterators

H.107.2.2 iterator_category

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::iterator_category
```

Initial value:

```
std::bidirectional_iterator_tag
```

Type of iterator

H.107.2.3 pointer

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::pointer = value_type*
```

Pointer to the type iterated over

H.107.2.4 reference

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::reference = value_type&
```

Reference to the type iterated over

H.107.2.5 value_type

```
template<class Key , class T >
using BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::value_type = std::pair<Key,
T>
```

Type when dereferencing iterators

H.107.3 Constructor & Destructor Documentation

H.107.3.1 OrderedMapIterator()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::OrderedMapIterator ()
```

Constructor

H.107.3.2 ~OrderedMapIterator()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::~ OrderedMapIterator ()
```

Destructor

H.107.4 Member Function Documentation

H.107.4.1 operator"!=()

```
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.

H.107.4.2 operator*()

```
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.

H.107.4.3 operator++() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T > &  BiometricEvaluation::Memory::↩
OrderedMapIterator< Key, T >::operator++ ()
```

Move to the next pair

H.107.4.4 operator++() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T >  BiometricEvaluation::Memory::Ordered↩
MapIterator< Key, T >::operator++ (
            int )
```

Move to the next pair

H.107.4.5 operator--() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T > &  BiometricEvaluation::Memory::↩
OrderedMapIterator< Key, T >::operator-- ()
```

Move to the previous pair.

H.107.4.6 operator--() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T >  BiometricEvaluation::Memory::Ordered↩
MapIterator< Key, T >::operator-- (
            int )
```

Move to the previous pair.

H.107.4.7 operator->()

```
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.

H.107.4.8 operator==()

```
template<class Key , class T >
bool  BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::operator== (
            const  OrderedMapIterator< Key, T > & rhs) const
```

Test for iterator equality.

Parameters

rhs	Object on the right-hand side of the expression.

Returns

Whether or not this iterator is equivalent to rhs.

H.108 BiometricEvaluation::Feature::AN2K11EFS::Orientation Struct Reference

Representation of orientation (deviation from upright) and its uncertainty.

`#include <be_feature_an2k11efs.h>`

Public Types

- enum class **EncodingMethod** { **Default** , **Indeterminate** , **UserDefined** }

Public Attributes

- **EncodingMethod encodingMethod**
- int **eod**
- bool **has_euc**
- int **euc**

Static Public Attributes

- static const int **EODDefault** = 0
- static const int **EUCDefault** = 15
- static const int **EUCIndeterminate** = 180

H.108.1 Detailed Description

Representation of orientation (deviation from upright) and its uncertainty.

H.108.2 Member Enumeration Documentation

H.108.2.1 EncodingMethod

`enum class` **`BiometricEvaluation::Feature::AN2K11EFS::Orientation::EncodingMethod`** `[strong]`

Interpretation of encoded orientation values.

Enumerator

Default	No orientation was encoded
Indeterminate	Encoded value indicates orientation was not determined.
UserDefined	Value was encoded

H.108.3 Member Data Documentation

H.108.3.1 encodingMethod

EncodingMethod BiometricEvaluation::Feature::AN2K11EFS::Orientation::encodingMethod

Interpretation of encoded values.

H.108.3.2 eod

int BiometricEvaluation::Feature::AN2K11EFS::Orientation::eod

Direction

H.108.3.3 EODDefault

const int BiometricEvaluation::Feature::AN2K11EFS::Orientation::EODDefault = 0 [static]

ANSI/NIST default direction

H.108.3.4 euc

int BiometricEvaluation::Feature::AN2K11EFS::Orientation::euc

Uncertainty

H.108.3.5 EUCDefault

const int BiometricEvaluation::Feature::AN2K11EFS::Orientation::EUCDefault = 15 [static]

ANSI/NIST default uncertainty

H.108.3.6 EUCIndeterminate

```
const int BiometricEvaluation::Feature::AN2K11EFS::Orientation::EUCIndeterminate = 180  [static]
```

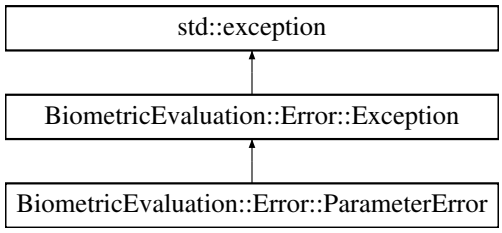
ANSI/NIST indeterminate uncertainty

H.109 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.109.1 Detailed Description

An invalid parameter was passed to a constructor or method.

H.109.2 Constructor & Destructor Documentation

H.109.2.1 ParameterError() [1/2]

```
BiometricEvaluation::Error::ParameterError::ParameterError ()
```

Construct a **ParameterError** (p. 655) object with the default information string.

H.109.2.2 ParameterError() [2/2]

```
BiometricEvaluation::Error::ParameterError::ParameterError (
            const std::string & info)
```

Construct a **ParameterError** (p. 655) object with an information string appended to the default information string.

H.110 BiometricEvaluation::Feature::AN2K11EFS::Pattern Struct Reference

`#include <be_feature_an2k11efs.h>`

Public Types

- enum class **GeneralClassification** {
  Arch , **Whorl** , **RightSlantLoop** , **LeftSlantLoop** ,
  Amputation , **TemporarilyUnavailable** , **Unclassifiable** , **Scar** ,
  DissociatedRidges }

  General pattern classification.
- enum class **ArchSubclassification** { **Plain** , **Tented** }
- enum class **WhorlSubclassification** { **Plain** , **CentralPocketLoop** , **DoubleLoop** , **Accidental** }
- enum class **WhorlDeltaRelationship** { **Inner** , **Outer** , **Meeting** }

Public Attributes

- bool **present** {false}
- **GeneralClassification general**
- bool **hasSubclass** {false}
- union {
  ArchSubclassification **arch**
  WhorlSubclassification **whorl**
  } **subclass**

- bool **hasWhorlDeltaRelationship** {false}
- **WhorlDeltaRelationship whorlDeltaRelationship**

H.110.1 Detailed Description

Fingerprint classification.

H.110.2 Member Enumeration Documentation

H.110.2.1 GeneralClassification

`enum class` **`BiometricEvaluation::Feature::AN2K11EFS::Pattern::GeneralClassification`** `[strong]`

General pattern classification.
@seealso **BiometricEvaluation::Finger::PatternClassification** (p. 124)

H.110.2.2 WhorlDeltaRelationship

`enum class` **`BiometricEvaluation::Feature::AN2K11EFS::Pattern::WhorlDeltaRelationship`** `[strong]`

Relationship between multiple deltas in a whorl

H.111 BiometricEvaluation::Feature::AN2K7Minutiae::Pattern↩ Classification Class Reference

Pattern classification codes.
`#include <be_feature_an2k7minutiae.h>`

Classes

- struct **Entry**

Public Types

- using **Entry** = struct Entry

H.111.1 Detailed Description

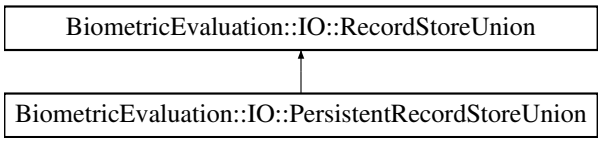
Pattern classification codes.

H.112 BiometricEvaluation::IO::PersistentRecordStoreUnion Class Reference

An implementation of **IO::RecordStoreUnion** (p. 729) that persists across instantiations.

`#include <be_io_persistentrecordstoreunion.h>`

Inheritance diagram for BiometricEvaluation::IO::PersistentRecordStoreUnion:



Public Member Functions

- **PersistentRecordStoreUnion** (const std::string &path)

  *Open an existing **PersistentRecordStoreUnion** (p. 657).*
- **PersistentRecordStoreUnion** (const std::string &path, const std::map< const std::string, const std↩ ::string > &recordStores)

  *Create a new **PersistentRecordStoreUnion** (p. 657).*
- **PersistentRecordStoreUnion** (const std::string &path, std::initializer_list< std::pair< const std::string, const std::string > > &recordStores)

  *Create a new **PersistentRecordStoreUnion** (p. 657).*
- ~**PersistentRecordStoreUnion** ()=default

Public Member Functions inherited from BiometricEvaluation::IO::RecordStoreUnion

- **RecordStoreUnion** (const std::map< const std::string, const std::string > &recordStores)
- **RecordStoreUnion** (std::map< const std::string, const std::string >::iterator first, std::map< const std↩ ::string, const std::string >::iterator last)
- **RecordStoreUnion** (std::initializer_list< std::pair< const std::string, const std::string > > record↩ Stores)
- **RecordStoreUnion** (const std::map< const std::string, const std::shared_ptr< **BiometricEvaluation**↩ **::IO::RecordStore** > > &recordStores)
- **RecordStoreUnion** (std::map< const std::string, const std::shared_ptr< **BiometricEvaluation::IO**↩ **::RecordStore** > >::iterator first, std::map< const std::string, const std::shared_ptr< **Biometric**↩ **Evaluation::IO::RecordStore** > >::iterator last)

- **RecordStoreUnion** (std::initializer_list< std::pair< const std::string, const std::shared_ptr< **BiometricEvaluation::IO::RecordStore** > > > recordStores)
- std::shared_ptr< **BiometricEvaluation::IO::RecordStore** > **getRecordStore** (const std::string &name) const

  *Obtain a pointer to an open **RecordStore*** (p. 700).
- std::vector< std::string > **getNames** () const

  Obtain the names of RecordStores set during construction.
- std::map< const std::string, **BiometricEvaluation::Memory::uint8Array** > **read** (const std::string &key) const

  Read a key from all member RecordStores.
- std::map< const std::string, uint64_t > **length** (const std::string &key) const

  Retrieve the length of a key from all member RecordStores.
- **RecordStoreUnion** (const **RecordStoreUnion** &)=delete
- **RecordStoreUnion** & **operator=** (const **RecordStoreUnion** &)=delete
- ~**RecordStoreUnion** ()

Additional Inherited Members

Protected Member Functions inherited from BiometricEvaluation::IO::RecordStoreUnion

- **RecordStoreUnion** ()

  Empty constructor for children.
- void **setImpl** (const std::shared_ptr< RecordStoreUnion::Impl > &pimpl)

  Change the implementation of this object.

H.112.1 Detailed Description

An implementation of **IO::RecordStoreUnion** (p. 729) that persists across instantiations.

State data is saved within the file system under the path specified during construction.

H.112.2 Constructor & Destructor Documentation

H.112.2.1 PersistentRecordStoreUnion() [1/3]

```
BiometricEvaluation::IO::PersistentRecordStoreUnion::PersistentRecordStoreUnion (
            const std::string & path)
```

Open an existing **PersistentRecordStoreUnion** (p. 657).

Parameters

path	Path at which **RecordStoreUnion** (p. 729) was persisted.

H.112.2.2 PersistentRecordStoreUnion() [2/3]

```
BiometricEvaluation::IO::PersistentRecordStoreUnion::PersistentRecordStoreUnion (
            const std::string & path,
            const std::map< const std::string, const std::string > & recordStores)
```

Create a new **PersistentRecordStoreUnion** (p. 657).

Parameters

path	Path at which **RecordStoreUnion** (p. 729) will be persisted.
recordStores	Initial RecordStores members of the union.

H.112.2.3 PersistentRecordStoreUnion() [3/3]

```
BiometricEvaluation::IO::PersistentRecordStoreUnion::PersistentRecordStoreUnion (
            const std::string & path,
            std::initializer_list< std::pair< const std::string, const std::string > > & recordStores)
```

Create a new **PersistentRecordStoreUnion** (p. 657).

Parameters

path	Path at which **RecordStoreUnion** (p. 729) will be persisted.

Parameters

mode	Mode in which to open RecordStores in the union.
recordStores	Initial RecordStores members of the union.

H.112.2.4 ~PersistentRecordStoreUnion()

`BiometricEvaluation::IO::PersistentRecordStoreUnion::~PersistentRecordStoreUnion () [default]`

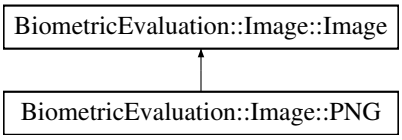
Destructor

H.113 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, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **PNG** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Memory::uint8Array getRawData** () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::uint8Array getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.

Public Member Functions inherited from BiometricEvaluation::Image::Image

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm** **getCompressionAlgorithm** () const

  Accessor for the CompressionAlgorithm of the image.
- **Resolution** **getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array** **getData** () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual **Memory::uint8Array** **getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Size** **getDimensions** () const

  Accessor for the dimensions of the image in pixels.
- uint32_t **getColorDepth** () const

  Accessor for the color depth of the image in bits.
- uint16_t **getBitDepth** () const

  Accessor for the number of bits per color component.
- bool **hasAlphaChannel** () const

  Accessor for the presence of an alpha channel.
- statusCallback_t **getStatusCallback** () const

  Get handle to status callback function.
- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Static Public Member Functions

- static bool **isPNG** (const uint8_t *data, uint64_t size)

Static Public Member Functions inherited from BiometricEvaluation::Image::Image

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.
- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

*Determine the image type of an image file and create an **Image** (p. 477) object.*

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm** **getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm** **getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.
- static **BiometricEvaluation::Image::Raw** **getRawImage** (const std::shared_ptr< **BiometricEvaluation::Image::Image** > &image)

  *Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).*
- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::Image::Image

- using **statusCallback_t**

Protected Member Functions inherited from BiometricEvaluation::Image::Image

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .
- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.
- void **setColorDepth** (const uint32_t colorDepth)

  Mutator for the color depth of the image in bits.
- void **setBitDepth** (const uint16_t bitDepth)

  Mutator for the number of bits per component for color components in the image, in bits.
- const uint8_t * **getDataPointer** () const
- uint64_t **getDataSize** () const
- void **setHasAlphaChannel** (const bool **hasAlphaChannel**)

  Mutator for the presence of an alpha channel.

H.113.1 Detailed Description

A PNG-encoded image.

H.113.2 Member Function Documentation

H.113.2.1 getRawData()

`Memory::uint8Array BiometricEvaluation::Image::PNG::getRawData () const [virtual]`

Accessor for the raw image data. The data returned should not be compressed or encoded.

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 486).

H.113.2.2 getRawGrayscaleData()

```
Memory::uint8Array BiometricEvaluation::Image::PNG::getRawGrayscaleData (
            uint8_t depth) const  [virtual]
```

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 16, 8, or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.
Error::NotImplemented *(p. 636)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 655)*	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implements **BiometricEvaluation::Image::Image** (p. 487).

H.113.2.3 isPNG()

```
static bool BiometricEvaluation::Image::PNG::isPNG (
            const uint8_t * data,
            uint64_t size)  [static]
```

Whether or not data is a **PNG** (p. 660) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a **PNG** (p. 660) image, false otherwise

H.114 BiometricEvaluation::Feature::Sort::Polar Class Reference

Sort (p. 117) by increasing distance from center and angle (theta).

`#include <be_feature_sort.h>`

Public Member Functions

- **Polar** (const **BiometricEvaluation::Image::Coordinate** &center)

  Polar *(p. 664) constructor.*
- bool **operator()** (const **BiometricEvaluation::Feature::MinutiaPoint** &lhs, const **BiometricEvaluation::Feature::MinutiaPoint** &rhs) const

Static Public Member Functions

- static **BiometricEvaluation::Image::Coordinate centerOfMinutiaeMass** (const BiometricEvaluation::Feature::MinutiaPointSet &mps)

  Obtain the center of minutiae mass.
- static **BiometricEvaluation::Image::Coordinate centerOfImage** (const **BiometricEvaluation::Image::Size** &size)

  Obtain the center point of an image.

H.114.1 Detailed Description

Sort (p. 117) by increasing distance from center and angle (theta).

H.114.2 Constructor & Destructor Documentation

H.114.2.1 Polar()

```
BiometricEvaluation::Feature::Sort::Polar::Polar (
            const  BiometricEvaluation::Image::Coordinate & center)
```

Polar (p. 664) constructor.

Parameters

center	Coordinate to use for center of image.

@seealso centerOfMinutiaeMass @seealso centerOfImage

H.114.3 Member Function Documentation

H.114.3.1 centerOfImage()

```
static  BiometricEvaluation::Image::Coordinate BiometricEvaluation::Feature::Sort::Polar::center↩
OfImage (
            const  BiometricEvaluation::Image::Size & size)  [static]
```

Obtain the center point of an image.

Parameters

size	Size of an image.

Note

If dimensions are odd, integer division is applied.

H.114.3.2 centerOfMinutiaeMass()

```
static  BiometricEvaluation::Image::Coordinate BiometricEvaluation::Feature::Sort::Polar::center↩
OfMinutiaeMass (
            const BiometricEvaluation::Feature::MinutiaPointSet & mps)  [static]
```

Obtain the center of minutiae mass.

Parameters

mps	Collection of minutia points.

Returns

Center of minutiae mass for mps.

Exceptions

Error::StrategyError *(p. 789)*	No minutia.

H.114.3.3 operator()()

```
bool BiometricEvaluation::Feature::Sort::Polar::operator() (
            const  BiometricEvaluation::Feature::MinutiaPoint & lhs,
            const  BiometricEvaluation::Feature::MinutiaPoint & rhs) const
```

MinutiaPoint (p. 619) polar ascending comparator.

H.115 BiometricEvaluation::Face::PoseAngle Struct Reference

Representation of pose angle and uncertainty.

`#include <be_face.h>`

Public Attributes

- uint8_t **yaw**
- uint8_t **pitch**
- uint8_t **roll**
- uint8_t **yawUncertainty**
- uint8_t **pitchUncertainty**
- uint8_t **rollUncertainty**

H.115.1 Detailed Description

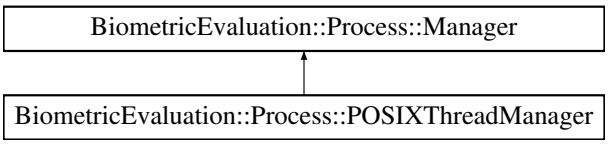
Representation of pose angle and uncertainty.

H.116 BiometricEvaluation::Process::POSIXThreadManager Class Reference

Manager (p. 596) implementation that starts Workers in POSIX threads.

`#include <be_process_posixthreadmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::POSIXThreadManager:



Public Member Functions

- **POSIXThreadManager** ()
- std::shared_ptr< **WorkerController** > **addWorker** (std::shared_ptr< **Worker** > worker)

  *Adds a **Worker** (p. 828) to be managed by this **Manager** (p. 596).*
- void **startWorkers** (bool wait=true, bool communicate=false)

  *Begin **Worker** (p. 828)'s work.*
- void **startWorker** (std::shared_ptr< **WorkerController** > worker, bool wait=true, bool communicate=false)

  *Start a **Worker** (p. 828).*
- void **stopWorker** (std::shared_ptr< **WorkerController** > workerController)

  *Ask **Worker** (p. 828) to exit.*
- void **waitForWorkerExit** ()

  Block until all Workers have exited.
- ~**POSIXThreadManager** ()

  ~POSIXThreadManager destructor.

Public Member Functions inherited from BiometricEvaluation::Process::Manager

- **Manager** ()

  ***Manager** (p. 596) constructor.*
- virtual uint32_t **getNumCompletedWorkers** () const

  Obtain the number of Workers that have exited.
- virtual uint32_t **getNumActiveWorkers** () const

  Obtain the number of Workers that are still working.
- virtual uint32_t **getTotalWorkers** () const

  Obtain the number of Workers this class is handling.
- virtual void **reset** ()

  Reuse all Workers.
- virtual bool **waitForMessage** (std::shared_ptr< **WorkerController** > &sender, int *nextFD=nullptr, int numSeconds=-1) const

  *Wait for a message from a **Worker** (p. 828).*
- virtual bool **getNextMessage** (std::shared_ptr< **WorkerController** > &sender, **Memory::uint8Array** &message, int numSeconds=-1) const

  *Obtain a message from a **Worker** (p. 828).*
- virtual void **broadcastMessage** (**Memory::uint8Array** &message) const

  Send one message to all Workers.
- virtual ~**Manager** ()

  ***Manager** (p. 596) destructor.*

Additional Inherited Members

Protected Attributes inherited from BiometricEvaluation::Process::Manager

- std::vector< std::shared_ptr< **WorkerController** > > **_workers**
- std::vector< std::shared_ptr< **WorkerController** > > **_pendingExit**

H.116.1 Detailed Description

Manager (p. 596) implementation that starts Workers in POSIX threads.

H.116.2 Constructor & Destructor Documentation

H.116.2.1 POSIXThreadManager()

```
BiometricEvaluation::Process::POSIXThreadManager::POSIXThreadManager ()
```

POSIXThreadManager (p. 666) constructor.

H.116.3 Member Function Documentation

H.116.3.1 addWorker()

```
std::shared_ptr< WorkerController > BiometricEvaluation::Process::POSIXThreadManager::addWorker (
            std::shared_ptr< Worker > worker) [virtual]
```

Adds a **Worker** (p. 828) to be managed by this **Manager** (p. 596).

Parameters

worker	A **Worker** (p. 828) instance to run.

Returns

shared_ptr to worker.

Implements **BiometricEvaluation::Process::Manager** (p. 597).

H.116.3.2 startWorker()

```
void BiometricEvaluation::Process::POSIXThreadManager::startWorker (
            std::shared_ptr< WorkerController > worker,
            bool wait = true,
            bool communicate = false) [virtual]
```

Start a **Worker** (p. 828).

Parameters

worker	Pointer to a **WorkerController** (p. 834) that is being managed by this **Manager** (p. 596) instance.
wait	Whether or not to wait for this **Worker** (p. 828) to exit before returning control to the caller.
communicate	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists (p. 637)	worker is already working.
Error::StrategyError (p. 789)	worker is not managed by this **Manager** (p. 596) instance.

Implements **BiometricEvaluation::Process::Manager** (p. 600).

H.116.3.3 startWorkers()

```
void BiometricEvaluation::Process::POSIXThreadManager::startWorkers (
            bool wait = true,
            bool communicate = false)  [virtual]
```

Begin **Worker** (p. 828)'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 (p. 637)	At least one **Worker** (p. 828) is already working.
Error::StrategyError (p. 789)	Problem starting the Workers.

Implements **BiometricEvaluation::Process::Manager** (p. 601).

H.116.3.4 stopWorker()

```
void BiometricEvaluation::Process::POSIXThreadManager::stopWorker (
            std::shared_ptr< WorkerController > workerController)  [virtual]
```

Ask **Worker** (p. 828) to exit.

Parameters

workerController	Pointer to the **Worker↩ Controller** (p. 834) that should be stopped.

Exceptions

Error::ObjectDoesNotExist (p. 637)	worker is not working.
Error::StrategyError (p. 789)	Problem sending the signal.

Implements **BiometricEvaluation::Process::Manager** (p. 602).

H.116.3.5 waitForWorkerExit()

```
void BiometricEvaluation::Process::POSIXThreadManager::waitForWorkerExit () [virtual]
```

Block until all Workers have exited.

Use this method if wait=false was set during a call to startWorker(s) but now wait=true is desired.

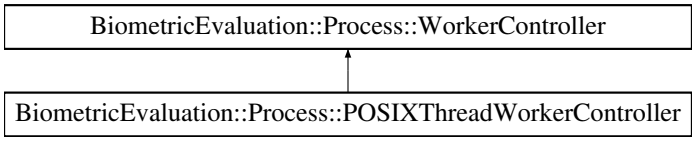
Implements **BiometricEvaluation::Process::Manager** (p. 604).

H.117 BiometricEvaluation::Process::POSIXThreadWorker↩ Controller Class Reference

Decorated **Worker** (p. 828) returned from a **Process::POSIXThreadManager** (p. 666).

`#include <be_process_posixthreadmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::POSIXThreadWorkerController:



Public Member Functions

- void **reset** ()

  *Reuse the **Worker*** (p. 828).

- bool **isWorking** () const

  *Obtain whether or not **Worker*** (p. 828) *is working.*

- bool **everWorked** () const

  *Obtain whether or not this **Worker*** (p. 828) *has ever worked.*

- ~**POSIXThreadWorkerController** ()

  POSIXThreadWorkerController *(p. 671) destructor.*

Public Member Functions inherited from BiometricEvaluation::Process::WorkerController

- **WorkerController** (std::shared_ptr< **Worker** > worker)
- virtual void **sendMessageToWorker** (const **Memory::uint8Array** &message)

  Send a message to the ***Worker*** *(p. 828) contained within this* ***WorkerController*** *(p. 834).*
- virtual void **setParameter** (const std::string &name, std::shared_ptr< void > argument)

  Set the parameter to be passed to the ***Worker*** *(p. 828).*
- virtual void **setParameterFromDouble** (const std::string &name, double argument)

  Set a double parameter to be passed to the ***Worker*** *(p. 828).*
- virtual void **setParameterFromInteger** (const std::string &name, int64_t argument)

  Set an integer parameter to be passed to the ***Worker*** *(p. 828).*
- virtual void **setParameterFromString** (const std::string &name, const std::string &argument)

  Set a string parameter to be passed to the ***Worker*** *(p. 828).*
- bool **finishedWorking** () const

  Obtain whether or not this ***Worker*** *(p. 828) has both started and finished its task.*
- std::shared_ptr< **Worker** > **getWorker** () const

  Obtain the ***Worker*** *(p. 828) instance being wrapped.*
- virtual int32_t **getExitStatus** () const final

  Obtain the exit status of the wrapped ***Worker*** *(p. 828).*
- virtual ~**WorkerController** ()

  WorkerController *(p. 834) destructor.*

Friends

- class **POSIXThreadManager**

Additional Inherited Members

Protected Attributes inherited from BiometricEvaluation::Process::WorkerController

- std::shared_ptr< **Worker** > **_worker**
- bool **_rvSet**
- int32_t **_rv**

H.117.1 Detailed Description

Decorated **Worker** (p. 828) returned from a **Process::POSIXThreadManager** (p. 666).

H.117.2 Member Function Documentation

H.117.2.1 everWorked()

`bool BiometricEvaluation::Process::POSIXThreadWorkerController::everWorked () const [virtual]`

Obtain whether or not this **Worker** (p. 828) has ever worked.

Returns

true the **Worker** (p. 828) has ever or is currently working, false otherwise.

Note

reset() (p. 673) will change the result of this method.

Implements **BiometricEvaluation::Process::WorkerController** (p. 836).

H.117.2.2 isWorking()

`bool BiometricEvaluation::Process::POSIXThreadWorkerController::isWorking () const [virtual]`

Obtain whether or not **Worker** (p. 828) is working.

Returns

Whether or not the **Worker** (p. 828) is working.

Implements **BiometricEvaluation::Process::WorkerController** (p. 837).

H.117.2.3 reset()

`void BiometricEvaluation::Process::POSIXThreadWorkerController::reset () [virtual]`

Reuse the **Worker** (p. 828).

Exceptions

Error::ObjectExists *(p. 637)*	The previously started **Worker** (p. 828) is still running.

Reimplemented from **BiometricEvaluation::Process::WorkerController** (p. 837).

H.118 BiometricEvaluation::View::AN2KViewVariableResolution::PrintPositionCoordinate Struct Reference

Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

`#include <be_view_an2kview_varres.h>`

Public Attributes

- **Finger::FingerImageCode fingerView**
- **Finger::FingerImageCode segment**
- Image::CoordinateSet **coordinates**

H.118.1 Detailed Description

Offsets to the bounding boxes for the EJI, full finger views, or EJI segments.

H.118.2 Member Data Documentation

H.118.2.1 coordinates

`Image::CoordinateSet BiometricEvaluation::View::AN2KViewVariableResolution::PrintPositionCoordinate::coordinates`

Two coordinates forming bounding box

H.118.2.2 fingerView

`Finger::FingerImageCode` `BiometricEvaluation::View::AN2KViewVariableResolution::PrintPositionCoordinate::fingerView`

Full finger view being bounded

H.118.2.3 segment

`Finger::FingerImageCode` `BiometricEvaluation::View::AN2KViewVariableResolution::PrintPositionCoordinate::segment`

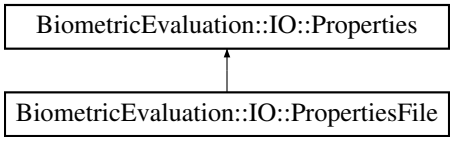
Segment within full finger view bound

H.119 BiometricEvaluation::IO::Properties Class Reference

Maintain key/value pairs of strings, with each property matched to one value.

`#include <be_io_properties.h>`

Inheritance diagram for BiometricEvaluation::IO::Properties:



Public Member Functions

- **Properties** (**IO::Mode** mode= **IO::Mode::ReadWrite**, const std::map< std::string, std::string > &defaults={})

  *Construct a new **Properties** (p. 674) object.*
- **Properties** (const uint8_t *buffer, const size_t size, **IO::Mode** mode= **IO::Mode::ReadWrite**, const std::map< std::string, std::string > &defaults={})

  *Construct a new **Properties** (p. 674) object from the contents of a buffer.*
- virtual void **setProperty** (const std::string &property, const std::string &value)

  Set a property with a value.
- virtual void **setPropertyFromInteger** (const std::string &property, int64_t value)

  Set a property with an integer value.
- virtual void **setPropertyFromDouble** (const std::string &property, double value)

  Set a property with a double value.
- virtual void **setPropertyFromBoolean** (const std::string &property, bool value)

  Set a property with a boolean value.
- virtual void **removeProperty** (const std::string &property)

  Remove a property.

- virtual std::string **getProperty** (const std::string &property) const

  Retrieve a property value as a string object.
- virtual int64_t **getPropertyAsInteger** (const std::string &property) const

  Retrieve a property value as an integer value.
- virtual double **getPropertyAsDouble** (const std::string &property) const

  Retrieve a property value as a double value.
- virtual bool **getPropertyAsBoolean** (const std::string &property) const
- std::vector< std::string > **getPropertyKeys** () const

  Retrieve a set of all property keys.
- virtual ~**Properties** ()

Protected Member Functions

- **BiometricEvaluation::IO::Mode getMode** () const

  *Obtain the mode of the **Properties** (p. 674) object.*
- void **initWithBuffer** (const **Memory::uint8Array** &buffer, const std::map< std::string, std::string > &defaults)

  Initialize the PropertiesMap with the contents of a properly formatted buffer.
- void **initWithBuffer** (const uint8_t *const buffer, size_t size, const std::map< std::string, std::string > &defaults)

  Initialize the PropertiesMap with the contents of a properly formatted buffer.

H.119.1 Detailed Description

Maintain key/value pairs of strings, with each property matched to one value.

H.119.2 Constructor & Destructor Documentation

H.119.2.1 Properties() [1/2]

```
BiometricEvaluation::IO::Properties::Properties (
                IO::Mode mode =  IO::Mode::ReadWrite,
              const std::map< std::string, std::string > & defaults = {})
```

Construct a new **Properties** (p. 674) object.

Parameters

in	*mode*	The read-/write mode of the object.
in	*defaults*	Default property-/-value pairs to insert.

H.119.2.2 Properties() [2/2]

```
BiometricEvaluation::IO::Properties::Properties (
            const uint8_t * buffer,
            const size_t size,
             IO::Mode mode =  IO::Mode::ReadWrite,
            const std::map< std::string, std::string > & defaults = {})
```

Construct a new **Properties** (p. 674) object from the contents of a buffer.

The format of the buffer can be seen in **PropertiesFile** (p. 683).

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.
in	*defaults*	Default property/-value pairs to insert.

Exceptions

Error::StrategyError *(p. 789)*	A line in the properties file is malformed.

H.119.2.3 ~Properties()

```
virtual BiometricEvaluation::IO::Properties::~Properties ()  [virtual]
```

Destructor

H.119.3 Member Function Documentation

H.119.3.1 getMode()

`BiometricEvaluation::IO::Mode BiometricEvaluation::IO::Properties::getMode () const [protected]`

Obtain the mode of the **Properties** (p. 674) object.

Returns

Mode (**Mode::ReadOnly** (p. **??**) or **Mode::ReadWrite** (p. **??**))

H.119.3.2 getProperty()

```
virtual std::string BiometricEvaluation::IO::Properties::getProperty (
            const std::string & property) const [virtual]
```

Retrieve a property value as a string object.

Parameters

in	*property*	The name of the property to get.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named property does not exist.

H.119.3.3 getPropertyAsDouble()

```
virtual double BiometricEvaluation::IO::Properties::getPropertyAsDouble (
            const std::string & property) const [virtual]
```

Retrieve a property value as a double value.

Parameters

in	*property*	The name of the property to get.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named property does not exist.
Error::ConversionError (p. 376)	The property value cannot be converted, due to non-numeric characters in the string, or the

H.119.3.4 getPropertyAsInteger()

```
virtual int64_t BiometricEvaluation::IO::Properties::getPropertyAsInteger (
            const std::string & property) const  [virtual]
```

Retrieve a property value as an integer value.

Integer value strings for properties can represent either decimal or hexadecimal values, which must be preceded with either "0x" or "0X".

Parameters

in	*property*	The name of the property to get.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named property does not exist.
Error::ConversionError (p. 376)	The property value cannot be converted, due to non-numeric characters in the string, or the

H.119.3.5 getPropertyKeys()

```
std::vector< std::string > BiometricEvaluation::IO::Properties::getPropertyKeys () const
```

Retrieve a set of all property keys.

Returns

A vector of property key strings.

H.119.3.6 initWithBuffer() [1/2]

```
void BiometricEvaluation::IO::Properties::initWithBuffer (
            const  Memory::uint8Array & buffer,
            const std::map< std::string, std::string > & defaults)  [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.
defaults	Default property/-value pairs.

Exceptions

Error::StrategyError *(p. 789)*	A line of the buffer is malformed.

H.119.3.7 initWithBuffer() [2/2]

```
void BiometricEvaluation::IO::Properties::initWithBuffer (
            const uint8_t *const buffer,
            size_t size,
            const std::map< std::string, std::string > & defaults)  [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.
defaults	Default property/-value pairs.

Exceptions

Error::StrategyError *(p. 789)*	A line of the buffer is malformed.

H.119.3.8 removeProperty()

```
virtual void BiometricEvaluation::IO::Properties::removeProperty (
            const std::string & property)  [virtual]
```

Remove a property.

Parameters

in	*property*	The name of the property to set.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The named property does not exist.
Error::StrategyError (p. 789)	The **Properties** (p. 674) object is read-only.

H.119.3.9 setProperty()

```
virtual void BiometricEvaluation::IO::Properties::setProperty (
            const std::string & property,
            const std::string & value)  [virtual]
```

Set a property with a value.

Both the property and value will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise, the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError (p. 789)	The **Properties** (p. 674) object is read-only.

H.119.3.10 setPropertyFromBoolean()

```
virtual void BiometricEvaluation::IO::Properties::setPropertyFromBoolean (
            const std::string & property,
            bool value)  [virtual]
```

Set a property with a boolean value.

The actual value to be written is implementation- defined and may not actually be preserved, but the boolean value is guaranteed to remain valid when read with getPropertyAsBoolean().

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError *(p. 789)*	The **Properties** (p. 674) object is read-only.

H.119.3.11 setPropertyFromDouble()

```
virtual void BiometricEvaluation::IO::Properties::setPropertyFromDouble (
            const std::string & property,
            double value)  [virtual]
```

Set a property with a double value.

The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError *(p. 789)*	The **Properties** (p. 674) object is read-only.

H.119.3.12 setPropertyFromInteger()

```
virtual void BiometricEvaluation::IO::Properties::setPropertyFromInteger (
            const std::string & property,
            int64_t value)  [virtual]
```

Set a property with an integer value.

The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

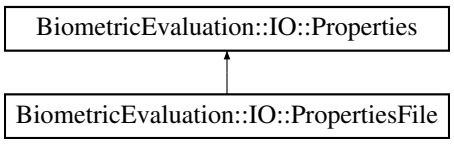
Error::StrategyError *(p. 789)*	The **Properties** (p. 674) object is read-only.

H.120 BiometricEvaluation::IO::PropertiesFile Class Reference

An **IO::Properties** (p. 674) object persisted in a file on disk.

`#include <be_io_propertiesfile.h>`

Inheritance diagram for BiometricEvaluation::IO::PropertiesFile:



Public Member Functions

- **PropertiesFile** (const std::string &pathname, **IO::Mode** mode= **IO::Mode::ReadOnly**, const std↩ ::map< std::string, std::string > &defaults={})

  *Construct a new **Properties** (p. 674) object from an existing or to be created properties file. The constructor will create the file when it does not exist.*
- void **sync** ()

  Write the properties to the underlying file, synchronizing the in-memory and on-disk versions.
- void **changeName** (const std::string &pathname)

  *Change the name of the **Properties** (p. 674), which means changing the name of the underlying file that stores the properties.*
- ~**PropertiesFile** ()
- **PropertiesFile** (const **PropertiesFile** &other)=delete

  Copy constructor (disabled).
- **PropertiesFile** & **operator=** (const **PropertiesFile** &other)=delete

  Assignment operator (disabled).

Public Member Functions inherited from BiometricEvaluation::IO::Properties

- **Properties** (**IO::Mode** mode= **IO::Mode::ReadWrite**, const std::map< std::string, std::string > &defaults={})

  *Construct a new **Properties** (p. 674) object.*
- **Properties** (const uint8_t *buffer, const size_t size, **IO::Mode** mode= **IO::Mode::ReadWrite**, const std::map< std::string, std::string > &defaults={})

  *Construct a new **Properties** (p. 674) object from the contents of a buffer.*
- virtual void **setProperty** (const std::string &property, const std::string &value)

  Set a property with a value.
- virtual void **setPropertyFromInteger** (const std::string &property, int64_t value)

  Set a property with an integer value.
- virtual void **setPropertyFromDouble** (const std::string &property, double value)

  Set a property with a double value.
- virtual void **setPropertyFromBoolean** (const std::string &property, bool value)

  Set a property with a boolean value.
- virtual void **removeProperty** (const std::string &property)

  Remove a property.
- virtual std::string **getProperty** (const std::string &property) const

  Retrieve a property value as a string object.
- virtual int64_t **getPropertyAsInteger** (const std::string &property) const

  Retrieve a property value as an integer value.
- virtual double **getPropertyAsDouble** (const std::string &property) const

  Retrieve a property value as a double value.
- virtual bool **getPropertyAsBoolean** (const std::string &property) const
- std::vector< std::string > **getPropertyKeys** () const

  Retrieve a set of all property keys.
- virtual ~**Properties** ()

Additional Inherited Members

Protected Member Functions inherited from BiometricEvaluation::IO::Properties

- **BiometricEvaluation::IO::Mode getMode** () const

  *Obtain the mode of the **Properties** (p. 674) object.*
- void **initWithBuffer** (const **Memory::uint8Array** &buffer, const std::map< std::string, std::string > &defaults)

  Initialize the PropertiesMap with the contents of a properly formatted buffer.
- void **initWithBuffer** (const uint8_t *const buffer, size_t size, const std::map< std::string, std::string > &defaults)

  Initialize the PropertiesMap with the contents of a properly formatted buffer.

H.120.1 Detailed Description

An **IO::Properties** (p. 674) object persisted in a file on disk.

An example file might look like this:

```
*     Name = John Smith
*     Age = 32
*     Favorite Hex Number = 0xffff
*
```

For property keys and values, leading and trailing whitespace is removed, therefore the call

```
props->setProperty("  My property   ", "   A Value  ");
```

results in an entry in the property file as

```
*     My property = A value
*
```

Therefore, the property names "Foo", " Foo", "Foo " are equivalent.

H.120.2 Constructor & Destructor Documentation

H.120.2.1 PropertiesFile() [1/2]

```
BiometricEvaluation::IO::PropertiesFile::PropertiesFile (
            const std::string & pathname,
             IO::Mode mode =  IO::Mode::ReadOnly,
            const std::map< std::string, std::string > & defaults = {})
```

Construct a new **Properties** (p. 674) object from an existing or to be created properties file. The constructor will create the file when it does not exist.

Parameters

in	*pathname*	The path to the file to store the properties.

Parameters

in	*mode*	The read-/write mode of the object.
in	*defaults*	Default property/-value pairs to insert.

Exceptions

Error::StrategyError (p. 789)	A line in the properties file is malformed.
Error::FileError (p. 420)	An error occurred when using the underlying storage system.

H.120.2.2 ~PropertiesFile()

```
BiometricEvaluation::IO::PropertiesFile::~PropertiesFile ()
```

Destructor

H.120.2.3 PropertiesFile() [2/2]

```
BiometricEvaluation::IO::PropertiesFile::PropertiesFile (
            const PropertiesFile & other) [delete]
```

Copy constructor (disabled).

Disabled because this object could represent a file on disk.

Parameters

other	**PropertiesFile** (p. 683) object to copy.

H.120.3 Member Function Documentation

H.120.3.1 changeName()

```
void BiometricEvaluation::IO::PropertiesFile::changeName (
            const std::string & pathname)
```

Change the name of the **Properties** (p. 674), which means changing the name of the underlying file that stores the properties.

Note

No check is made that the file is writeable at this time.

Parameters

in	*pathname*	The path to the **Properties** (p. 674) file.

Exceptions

Error::StrategyError *(p. 789)*	The object is read-only.
Error::ObjectExists *(p. 637)*	A file at pathname already exists.

H.120.3.2 operator=()

```
PropertiesFile & BiometricEvaluation::IO::PropertiesFile::operator= (
            const PropertiesFile & other) [delete]
```

Assignment operator (disabled).

Disabled because this object could represent a file on disk.

Parameters

other	**PropertiesFile** (p. 683) object to assign;

Returns

This **PropertiesFile** (p. 683) object, now containing the contents of other.

H.120.3.3 sync()

```
void BiometricEvaluation::IO::PropertiesFile::sync ()
```

Write the properties to the underlying file, synchronizing the in-memory and on-disk versions.

Exceptions

Error::FileError *(p. 420)*	An error occurred when using the underlying storage system.
Error::StrategyError *(p. 789)*	The object was constructed with nullptr as the file name, or is read-only.

H.121 BiometricEvaluation::Feature::Sort::Quality Class Reference

`#include <be_feature_sort.h>`

Public Member Functions

- bool **operator()** (const **BiometricEvaluation::Feature::MinutiaPoint** &lhs, const **BiometricEvaluation::Feature::MinutiaPoint** &rhs) const

  MinutiaPoint *(p. 619) quality ascending comparator.*

H.121.1 Detailed Description

Sort (p. 117) by increasing minutiae quality

H.122 BiometricEvaluation::Iris::INCITSView::QualitySubBlock Struct Reference

Representation of an iris quality block.

`#include <be_iris_incitsview.h>`

Public Attributes

- uint8_t **score**
- uint16_t **vendorID**
- uint16_t **algorithmID**

H.122.1 Detailed Description

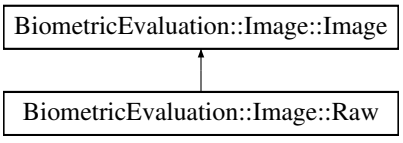
Representation of an iris quality block.

H.123 BiometricEvaluation::Image::Raw Class Reference

An image with no encoding or compression.

`#include <be_image_raw.h>`

Inheritance diagram for BiometricEvaluation::Image::Raw:



Public Member Functions

- **Raw** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Raw** (const **BiometricEvaluation::Memory::uint8Array** &data, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Memory::uint8Array getRawData** () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::uint8Array getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.

Public Member Functions inherited from BiometricEvaluation::Image::Image

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm getCompressionAlgorithm** () const

  Accessor for the CompressionAlgorithm of the image.
- **Resolution getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array getData** () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual **Memory::uint8Array getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Size getDimensions** () const

  Accessor for the dimensions of the image in pixels.
- uint32_t **getColorDepth** () const

  Accessor for the color depth of the image in bits.
- uint16_t **getBitDepth** () const

  Accessor for the number of bits per color component.
- bool **hasAlphaChannel** () const

  Accessor for the presence of an alpha channel.
- statusCallback_t **getStatusCallback** () const

  Get handle to status callback function.
- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::Image::Image

- using **statusCallback_t**

Static Public Member Functions inherited from BiometricEvaluation::Image::Image

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.
- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*
- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of an image file and create an **Image** (p. 477) object.*
- static **CompressionAlgorithm** **getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm** **getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.
- static **CompressionAlgorithm** **getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.
- static **BiometricEvaluation::Image::Raw** **getRawImage** (const std::shared_ptr< **BiometricEvaluation::Image::Image** > &image)

  *Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).*
- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Protected Member Functions inherited from BiometricEvaluation::Image::Image

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .
- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.
- void **setColorDepth** (const uint32_t colorDepth)

  Mutator for the color depth of the image in bits.
- void **setBitDepth** (const uint16_t bitDepth)

  Mutator for the number of bits per component for color components in the image, in bits.
- const uint8_t * **getDataPointer** () const
- uint64_t **getDataSize** () const
- void **setHasAlphaChannel** (const bool **hasAlphaChannel**)

  Mutator for the presence of an alpha channel.

H.123.1 Detailed Description

An image with no encoding or compression.

H.123.2 Member Function Documentation

H.123.2.1 getRawData()

`Memory::uint8Array BiometricEvaluation::Image::Raw::getRawData () const [virtual]`

Accessor for the raw image data. The data returned should not be compressed or encoded.

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 486).

H.123.2.2 getRawGrayscaleData()

`Memory::uint8Array BiometricEvaluation::Image::Raw::getRawGrayscaleData ( uint8_t depth) const [virtual]`

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 16, 8, or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.
Error::NotImplemented *(p. 636)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 655)*	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implements **BiometricEvaluation::Image::Image** (p. 487).

H.124 BiometricEvaluation::MPI::Receiver Class Reference

A class to represent an **MPI** (p. 162) task that receives WorkPackages containers from the **Distributor** (p. 405).

`#include <be_mpi_receiver.h>`

Public Member Functions

- **Receiver** (const std::string &propertiesFileName, const std::shared_ptr< **BiometricEvaluation::↩ MPI::WorkPackageProcessor** > &workPackageProcessor)

  Construct a new work package receiver.
- void **start** ()

  Start the receiving task.

H.124.1 Detailed Description

A class to represent an **MPI** (p. 162) task that receives WorkPackages containers from the **Distributor** (p. 405).

A receiver object depends on a set of properties contained in a file. The properties specify **MPI** (p. 162) settings, and other items. Subclasses of the class can add new properties.

Each receiver object is responsible for 1..n worker processes that are started when **Receiver::start()** (p. 693) is called. The receiver will start workers only when the distributor indicates that it has started successfully. Otherwise, the **Receiver** (p. 692) transitions to the shutdown state.

One of the optional properties is a Uniform Resource Locator (URL) for the Logsheet. If this property does not exist, no logging takes place (although applications can create their own Logsheet). If the URL is present, the framework will log at various points of processing. In the case of a FileLogsheet the framework will create more than one log file, each named after the ID of the **MPI** (p. 162) task created by the **MPI** (p. 162) runtime, and the child process created by **Receiver** (p. 692).

See also

IO::Properties (p. 674)

IO::Logsheet (p. 585)

MPI::Distributor (p. 405)

Process::Worker (p. 828)

H.124.2 Constructor & Destructor Documentation

H.124.2.1 Receiver()

```
BiometricEvaluation::MPI::Receiver::Receiver (
            const std::string & propertiesFileName,
            const std::shared_ptr< BiometricEvaluation::MPI::WorkPackageProcessor > & work↩
PackageProcessor)
```

Construct a new work package receiver.

Parameters

in	*propertiesFileName*	The name of the file containing the properties used by the receiver object.
in	*workPackageProcessor*	The object that will process the work received by this object.

Exceptions

Error::Exception (p. 412)	An error occurred when constructing this object.

H.124.3 Member Function Documentation

H.124.3.1 start()

```
void BiometricEvaluation::MPI::Receiver::start ()
```

Start the receiving task.

Upon starting, the **Receiver** (p. 692) object will begin communicating with the **Distributor** (p. 405) using **MPI** (p. 162) messages. This **Receiver** (p. 692) object will send a status message back to the **Distributor** (p. 405) indicating success or failure to initialize. Success includes the startup of at least one worker process.

H.125 BiometricEvaluation::IO::RecordStore::Record Struct Reference

Public Member Functions

- **Record** ()
- **Record** (const std::string &key, const **Memory::uint8Array** &data)

  *Create a **Record** (p. 694) from the key and data.*

Public Attributes

- std::string **key**
- **Memory::uint8Array data**

H.125.1 Constructor & Destructor Documentation

H.125.1.1 Record() [1/2]

```
BiometricEvaluation::IO::RecordStore::Record::Record ()
```

Default constructor.

H.125.1.2 Record() [2/2]

```
BiometricEvaluation::IO::RecordStore::Record::Record (
            const std::string & key,
            const  Memory::uint8Array & data)
```

Create a **Record** (p. 694) from the key and data.

Parameters

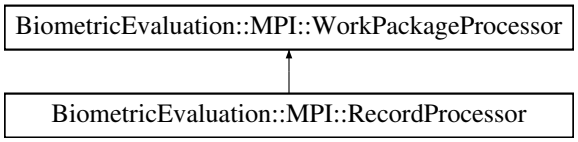
in	*key*	The record's key.
in	*data*	The record's data (value).

H.126 BiometricEvaluation::MPI::RecordProcessor Class Reference

An implementation of a work package processor that will extract record store keys, and optionally, values, from a **WorkPackage** (p. 841).

`#include <be_mpi_recordprocessor.h>`

Inheritance diagram for BiometricEvaluation::MPI::RecordProcessor:



Public Member Functions

- **RecordProcessor** (const std::string &propertiesFileName)

  Construct a work package processor with the given properties.
- virtual void **processRecord** (const std::string &key)=0

  Method implemented by child classes to perform an action using each record from the Record Store.
- virtual void **processRecord** (const std::string &key, const **Memory::uint8Array** &value)=0

  Method implemented by child classes to perform an action using each record from the Record Store.
- virtual std::shared_ptr< **WorkPackageProcessor** > **newProcessor** (std::shared_ptr< **IO::Logsheet** > &logsheet)=0

  Obtain an object that will process work packages. This method is part of the factory personality.
- virtual void **performInitialization** (std::shared_ptr< **IO::Logsheet** > &logsheet)=0

  Initialization function to be called before work is distributed to the work package processor.
- void **processWorkPackage** (**MPI::WorkPackage** &workPackage)

  Process (p. 170) *the data contents of the work package. This method is part of the worker personality.*

Public Member Functions inherited from BiometricEvaluation::MPI::WorkPackageProcessor

- virtual void **performShutdown** ()

  Terminiation function to be called during shut down after all work package processing is done.
- void **setLogsheet** (std::shared_ptr< **IO::Logsheet** > &logsheet)

  *Set the **IO::Logsheet*** (p. 585) *object that can be used to save message for objects of this class.*
- std::shared_ptr< **IO::Logsheet** > **getLogsheet** ()

  *Obtain the **IO::Logsheet*** (p. 585) *object that can be used to save message for objects of this class.*

Protected Member Functions

- std::shared_ptr< **MPI::RecordStoreResources** > **getResources** ()

H.126.1 Detailed Description

An implementation of a work package processor that will extract record store keys, and optionally, values, from a **WorkPackage** (p. 841).

Subclasses of this abstract class must implement the method to process the records associated with the keys.

H.126.2 Constructor & Destructor Documentation

H.126.2.1 RecordProcessor()

```
BiometricEvaluation::MPI::RecordProcessor::RecordProcessor (
            const std::string & propertiesFileName)
```

Construct a work package processor with the given properties.

A record processor uses a named record store to retrieve the data to be processed when only the key is delivered as part of a work package. When both key and value are part of the work package, there is no need to have access to the source record store.

Note

The size of a single value item is limited to 2^32 octets. If the size of the value item is larger, behavior is undefined.

Parameters

in	*propertiesFileName*	The name of the file containing the properties for this object.

Exceptions

Error::Exception (p. 412)	An error occurred, usually due to missing or incorrect properties.

H.126.3 Member Function Documentation

H.126.3.1 newProcessor()

```
virtual std::shared_ptr< WorkPackageProcessor > BiometricEvaluation::MPI::RecordProcessor↩
::newProcessor (
            std::shared_ptr< IO::Logsheet > & logsheet) [pure virtual]
```

Obtain an object that will process work packages. This method is part of the factory personality.

Parameters

logsheet	A shared pointer to the **IO::Logsheet** (p. 585) that may be used to save messages generated by the object.

Returns

A shared pointer to the work package processor.

Note

This method should always create a non-null **WorkPackageProcessor** (p. 843). If an error occurs during construction, throw a **Error::Exception** (p. 412) with a message to be caught and logged.

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 844).

H.126.3.2 performInitialization()

```
virtual void BiometricEvaluation::MPI::RecordProcessor::performInitialization (
            std::shared_ptr< IO::Logsheet > & logsheet) [pure virtual]
```

Initialization function to be called before work is distributed to the work package processor.

Implementations of this class can use this function to do any processing necessary before work is given to the processor, pre-forking.

This method is part of the factory personality. All state that is to be common across all package processor objects can be initialized in this method.

Parameters

logsheet	A shared pointer to the **IO::Logsheet** (p. 585) that may be used to save messages generated by the object.

Exceptions

Error::Exception *(p. 412)*	An implementation specific error occurred. The exception string will be logged by the **Framework** (

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 845).

H.126.3.3 processRecord() [1/2]

```
virtual void BiometricEvaluation::MPI::RecordProcessor::processRecord (
            const std::string & key)  [pure virtual]
```

Method implemented by child classes to perform an action using each record from the Record Store.

The source RecordStore must be accessible to the the implementation as the value for each key is not included.

Parameters

in	*key*	The key associated with the record that is to be processed.

Exceptions

Error::Exception *(p. 412)*	An error occurred processing the record: Missing record, input/output error, or memory allocation.

H.126.3.4 processRecord() [2/2]

```
virtual void BiometricEvaluation::MPI::RecordProcessor::processRecord (
            const std::string & key,
            const  Memory::uint8Array & value)  [pure virtual]
```

Method implemented by child classes to perform an action using each record from the Record Store.

Parameters

in	*key*	The key associated with the record that is to be processed.
in	*value*	The data from the record that is to be processed.

Exceptions

Error::Exception *(p. 412)*	An fatal error occurred when processing the work package; the processing responsible for this object

H.126.3.5 processWorkPackage()

```
void BiometricEvaluation::MPI::RecordProcessor::processWorkPackage (
              MPI::WorkPackage & workPackage)  [virtual]
```

Process (p. 170) the data contents of the work package. This method is part of the worker personality.

Parameters

in	*workPackage*	The work package.

Exceptions

Error::Exception (p. *412*)	An fatal error occurred when processing the work package; the processing responsible for this object

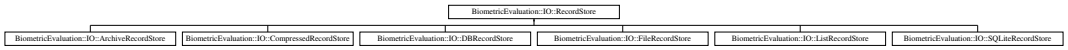
Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 846).

H.127 BiometricEvaluation::IO::RecordStore Class Reference

A class to represent a data storage mechanism.

`#include <be_io_recordstore.h>`

Inheritance diagram for BiometricEvaluation::IO::RecordStore:



Classes

- struct **Record**

Public Types

- enum class **Kind** { **BerkeleyDB** , **Archive** , **File** , **SQLite** , **Compressed** , **List** , **Default** = BerkeleyDB }
- using **Record** = struct Record
- using **iterator** = **IO::RecordStoreIterator**

Public Member Functions

- virtual std::string **getDescription** () const =0
- virtual unsigned int **getCount** () const =0
- virtual std::string **getPathname** () const =0
- virtual void **move** (const std::string &pathname)=0

  *Move the **RecordStore*** (p. *700*).
- virtual void **changeDescription** (const std::string &description)=0
- virtual uint64_t **getSpaceUsed** () const =0

  Obtain real storage utilization.
- virtual void **sync** () const =0
- virtual void **insert** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **insert** (const std::string &key, const void *const data, const uint64_t size)=0
- virtual void **remove** (const std::string &key)=0

- virtual **Memory::uint8Array read** (const std::string &key) const =0

  Read a complete record from a store.
- virtual void **replace** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const void *const data, const uint64_t size)
- virtual uint64_t **length** (const std::string &key) const =0
- virtual void **flush** (const std::string &key) const =0
- virtual **RecordStore::Record sequence** (int cursor= **BE_RECSTORE_SEQ_NEXT**)=0

  *Sequence through a **RecordStore** (p. 700), returning the key/data pairs.*
- virtual std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**)=0

  *Sequence through a **RecordStore** (p. 700), returning the key.*
- virtual void **setCursorAtKey** (const std::string &key)=0
- virtual bool **containsKey** (const std::string &key) const

  *Determines whether the **RecordStore** (p. 700) contains an element with the specified key.*
- virtual **iterator begin** () noexcept
- virtual **iterator end** () noexcept

Static Public Member Functions

- static bool **isRecordStore** (const std::string &pathname)

  *Determine if a location appears to be a **RecordStore** (p. 700).*
- static std::shared_ptr< **RecordStore** > **openRecordStore** (const std::string &pathname, **IO::Mode** mode= **Mode::ReadOnly**)

  *Open an existing **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static std::shared_ptr< **RecordStore** > **createRecordStore** (const std::string &pathname, const std::string &description, const **IO::RecordStore::Kind** &kind)

  *Create a new **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static void **removeRecordStore** (const std::string &pathname)
- static void **mergeRecordStores** (const std::string &mergePathname, const std::string &description, const **IO::RecordStore::Kind** &kind, const std::vector< std::string > &pathnames, const std::function< bool()> &interrupt=[ ]() {return(false);})

  *Create a new **RecordStore** (p. 700) that contains the contents of several other RecordStores.*

Static Public Attributes

- static const std::string **INVALIDKEYCHARS**
- static const int **BE_RECSTORE_SEQ_START** = 1
- static const int **BE_RECSTORE_SEQ_NEXT** = 2

H.127.1 Detailed Description

A class to represent a data storage mechanism.

A **RecordStore** (p. 700) is an abstraction that associates keys with a specific data item. Implementations of this abstraction can store the records in any format supported by the operating system, such as files or databases, rooted in the file system.

Certain characters are prohibited in the key string. See **IO::RecordStore::INVALIDKEYCHARS** (p. 718). A key string cannot begin with the space character.

See also

IO::ArchiveRecordStore (p. 292), **IO::DBRecordStore** (p. 391), **IO::FileRecordStore** (p. 433).

H.127.2 Member Enumeration Documentation

H.127.2.1 Kind

`enum class` **`BiometricEvaluation::IO::RecordStore::Kind`** `[strong]`

Possible types of **RecordStore** (p. 700)

Enumerator

BerkeleyDB	**DBRecord↩ Store** (p. 391)
Archive	**Archive↩ Record↩ Store** (p. 292)
File	**File↩ Record↩ Store** (p. 433)
SQLite	**SQLite↩ Record↩ Store** (p. 769)
Compressed	**Compressed↩ Record↩ Store** (p. 345)
List	**List↩ Record↩ Store** (p. 574)
Default	"↩ Default" **Record↩ Store** (p. 700) kind

H.127.3 Member Function Documentation

H.127.3.1 begin()

`virtual` **`iterator`** `BiometricEvaluation::IO::RecordStore::begin ()` `[virtual], [noexcept]`

Returns

Iterator to the first record.

H.127.3.2 changeDescription()

```
virtual void BiometricEvaluation::IO::RecordStore::changeDescription (
            const std::string & description)  [pure virtual]
```

Change the description of the **RecordStore** (p. 700).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 295), **BiometricEvaluation::IO::CompressedRecordStore** (p. 350), **BiometricEvaluation::IO::DBRecordStore** (p. 395), **BiometricEvaluation::IO::FileRecordStore** (p. 437), **BiometricEvaluation::IO::ListRecordStore** (p. 576), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 771).

H.127.3.3 containsKey()

```
virtual bool BiometricEvaluation::IO::RecordStore::containsKey (
            const std::string & key) const  [virtual]
```

Determines whether the **RecordStore** (p. 700) contains an element with the specified key.

Parameters

key	The key to locate.

Returns

True if the **RecordStore** (p. 700) contains an element with the key, false otherwise.

H.127.3.4 createRecordStore()

```
static std::shared_ptr<  RecordStore > BiometricEvaluation::IO::RecordStore::createRecordStore
(
            const std::string & pathname,
            const std::string & description,
            const  IO::RecordStore::Kind & kind)  [static]
```

Create a new **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

in	*pathname*	The directory of the store to be created.
in	*description*	The description of the store to be created.
in	*kind*	The kind of **RecordStore** (p. 700) to be created.

Returns

An managed pointer to the object representing the created store.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	The **RecordStore** (p. 700) does not exist.
Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

H.127.3.5 end()

```
virtual iterator BiometricEvaluation::IO::RecordStore::end () [virtual], [noexcept]
```

Returns

Iterator past the last record.

H.127.3.6 flush()

```
virtual void BiometricEvaluation::IO::RecordStore::flush (
            const std::string & key) const [pure virtual]
```

Commit the record's data to storage.

Parameters

in	*key*	The key of the record to be flushed.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 296), **BiometricEvaluation::IO::CompressedRecordStore** (p. 351), **BiometricEvaluation::IO::DBRecordStore** (p. 395), **BiometricEvaluation::IO::FileRecordStore** (p. 437), **BiometricEvaluation::IO::ListRecordStore** (p. 577), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 771).

H.127.3.7 getCount()

```
virtual unsigned int BiometricEvaluation::IO::RecordStore::getCount () const  [pure virtual]
```

Obtain the number of items in the **RecordStore** (p. 700).

Returns

The number of items in the **RecordStore** (p. 700).

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 296), **BiometricEvaluation::IO::CompressedRecordStore** (p. 351), **BiometricEvaluation::IO::DBRecordStore** (p. 395), **BiometricEvaluation::IO::FileRecordStore** (p. 437), **BiometricEvaluation::IO::ListRecordStore** (p. 577), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 772).

H.127.3.8 getDescription()

```
virtual std::string BiometricEvaluation::IO::RecordStore::getDescription () const  [pure virtual]
```

Obtain a textual description of the **RecordStore** (p. 700).

Returns

The **RecordStore** (p. 700)'s description.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 296), **BiometricEvaluation::IO::CompressedRecordStore** (p. 351), **BiometricEvaluation::IO::DBRecordStore** (p. 396), **BiometricEvaluation::IO::FileRecordStore** (p. 438), **BiometricEvaluation::IO::ListRecordStore** (p. 577), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 772).

H.127.3.9 getPathname()

```
virtual std::string BiometricEvaluation::IO::RecordStore::getPathname () const  [pure virtual]
```

Return the path name of the **RecordStore** (p. 700).

Returns

Where in the file system the **RecordStore** (p. 700) is located.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 297), **BiometricEvaluation::IO::CompressedRecordStore** (p. 352), **BiometricEvaluation::IO::DBRecordStore** (p. 396), **BiometricEvaluation::IO::FileRecordStore** (p. 438), **BiometricEvaluation::IO::ListRecordStore** (p. 578), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 772).

H.127.3.10 getSpaceUsed()

```
virtual uint64_t BiometricEvaluation::IO::RecordStore::getSpaceUsed () const  [pure virtual]
```

Obtain real storage utilization.

The amount of disk space used, for example. This is the actual space allocated by the underlying storage mechanism, in bytes.

Returns

The amount of backing storage used by the **RecordStore** (p. 700).

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 297), **BiometricEvaluation::IO::CompressedRecordStore** (p. 352), **BiometricEvaluation::IO::DBRecordStore** (p. 396), **BiometricEvaluation::IO::FileRecordStore** (p. 438), **BiometricEvaluation::IO::ListRecordStore** (p. 578), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 772).

H.127.3.11 insert() [1/2]

```
virtual void BiometricEvaluation::IO::RecordStore::insert (
            const std::string & key,
            const  Memory::uint8Array & data)  [virtual]
```

Insert a record into the store.

Parameters

in	*key*	The key of the record to be inserted.
in	*data*	The data for the record.

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Reimplemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 297), **BiometricEvaluation::IO::CompressedRecordStore** (p. 352), **BiometricEvaluation::IO::DBRecordStore** (p. 396), **BiometricEvaluation::IO::FileRecordStore** (p. 438), **BiometricEvaluation::IO::ListRecordStore** (p. 578), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 773).

H.127.3.12 insert() [2/2]

```
virtual void BiometricEvaluation::IO::RecordStore::insert (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [pure virtual]
```

Insert a record into the store.

Parameters

in	*key*	The key of the record to be inserted.
in	*data*	The data for the record.
in	*size*	The size of the record, in bytes.

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 298), **BiometricEvaluation::IO::CompressedRecordStore** (p. 353), **BiometricEvaluation::IO::DBRecordStore** (p. 397), **BiometricEvaluation::IO::FileRecordStore** (p. 439), **BiometricEvaluation::IO::ListRecordStore** (p. 579), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 773).

H.127.3.13 isRecordStore()

```
static bool BiometricEvaluation::IO::RecordStore::isRecordStore (
            const std::string & pathname)  [static]
```

Determine if a location appears to be a **RecordStore** (p. 700).

Parameters

pathname	The path name of the location to check.

Returns

true if `pathname` appears to point to a **RecordStore** (p. 700), false otherwise.

H.127.3.14 length()

```
virtual uint64_t BiometricEvaluation::IO::RecordStore::length (
            const std::string & key) const  [pure virtual]
```

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 299), **BiometricEvaluation::IO::CompressedRecordStore** (p. 353), **BiometricEvaluation::IO::DBRecordStore** (p. 398), **BiometricEvaluation::IO::FileRecordStore** (p. 440), **BiometricEvaluation::IO::ListRecordStore** (p. 579), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 774).

H.127.3.15 mergeRecordStores()

```
static void BiometricEvaluation::IO::RecordStore::mergeRecordStores (
            const std::string & mergePathname,
            const std::string & description,
            const  IO::RecordStore::Kind & kind,
            const std::vector< std::string > & pathnames,
            const std::function< bool()> & interrupt = []() {return(false);})  [static]
```

Create a new **RecordStore** (p. 700) that contains the contents of several other RecordStores.

Parameters

in	*mergePathname*	The path name of the new **RecordStore** (p. 700) that will be created.
in	*description*	The text used to describe the new **RecordStore** (p. 700).
in	*kind*	The kind of the new, merged **RecordStore** (p. 700).

Parameters

in	*pathnames*	Vector of path names to RecordStores to open. These are the RecordStores that will be merged to create the new **RecordStore** (p. 700).
in	*interrupt*	A function to be called during long operations to determine whether to interrupt and return.

Exceptions

Error::ObjectExists *(p. 637)*	A **RecordStore** (p. 700) at mergePathname already exists.
Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

H.127.3.16 move()

```
virtual void BiometricEvaluation::IO::RecordStore::move (
            const std::string & pathname)  [pure virtual]
```

Move the **RecordStore** (p. 700).
The **RecordStore** (p. 700) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 700).

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 299), **BiometricEvaluation::IO::CompressedRecordStore** (p. 354), **BiometricEvaluation::IO::DBRecordStore** (p. 398), **BiometricEvaluation::IO::FileRecordStore** (p. 440), **BiometricEvaluation::IO::ListRecordStore** (p. 580), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 774).

H.127.3.17 openRecordStore()

```
static std::shared_ptr< RecordStore > BiometricEvaluation::IO::RecordStore::openRecordStore
(
            const std::string & pathname,
             IO::Mode mode = Mode::ReadOnly)  [static]
```

Open an existing **RecordStore** (p. 700) 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** (p. 700) it is.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

in	*pathname*	The path name of the store to be opened.

Parameters

in	*mode*	The type of access a client of this **RecordStore** (p. 700) has.

Returns

An object representing the existing store.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The **RecordStore** (p. 700) does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

H.127.3.18 read()

```
virtual Memory::uint8Array BiometricEvaluation::IO::RecordStore::read (
            const std::string & key) const [pure virtual]
```

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

in	*key*	The key of the record to be read.

Returns

The record associated with the key.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 300), **BiometricEvaluation::IO::CompressedRecordStore** (p. 355), **BiometricEvaluation::IO::DBRecordStore** (p. 399), **BiometricEvaluation::IO::FileRecordStore** (p. 441), **BiometricEvaluation::IO::ListRecordStore** (p. 580), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 775).

H.127.3.19 remove()

```
virtual void BiometricEvaluation::IO::RecordStore::remove (
            const std::string & key)  [pure virtual]
```

Remove a record from the store.

Parameters

in	*key*	The key of the record to be removed.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 301), **BiometricEvaluation::IO::CompressedRecordStore** (p. 355), **BiometricEvaluation::IO::DBRecordStore** (p. 399), **BiometricEvaluation::IO::FileRecordStore** (p. 441), **BiometricEvaluation::IO::ListRecordStore** (p. 581), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 775).

H.127.3.20 removeRecordStore()

```
static void BiometricEvaluation::IO::RecordStore::removeRecordStore (
            const std::string & pathname)  [static]
```

Remove a **RecordStore** (p. 700) by deleting all persistant data associated with the store.

Parameters

in	*pathname*	The name of the existing **RecordStore** (p. 700).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record with the given key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

H.127.3.21 replace() [1/2]

```
virtual void BiometricEvaluation::IO::RecordStore::replace (
            const std::string & key,
            const  Memory::uint8Array & data)  [virtual]
```

Replace a complete record in a **RecordStore** (p. 700).

Parameters

in	*key*	The key of the record to be replaced.
in	*data*	The data for the record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 301), **BiometricEvaluation::IO::CompressedRecordStore** (p. 356), **BiometricEvaluation::IO::DBRecordStore** (p. 400), **BiometricEvaluation::IO::FileRecordStore** (p. 442), **BiometricEvaluation::IO::ListRecordStore** (p. 581), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 776).

H.127.3.22 replace() [2/2]

```
virtual void BiometricEvaluation::IO::RecordStore::replace (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [virtual]
```

Replace a complete record in a **RecordStore** (p. 700).

Parameters

in	*key*	The key of the record to be replaced.
in	*data*	The data for the record.

Parameters

in	*size*	The size of the record, in bytes.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	A record for the key does not exist.
Error::StrategyError *(p. 789)*	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 302), **BiometricEvaluation::IO::CompressedRecordStore** (p. 357), **BiometricEvaluation::IO::DBRecordStore** (p. 400), **BiometricEvaluation::IO::FileRecordStore** (p. 442), **BiometricEvaluation::IO::ListRecordStore** (p. 582), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 777).

H.127.3.23 sequence()

```
virtual RecordStore::Record BiometricEvaluation::IO::RecordStore::sequence (
            int cursor = BE_RECSTORE_SEQ_NEXT) [pure virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the function to return the next record. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

in	*cursor*	The location within the sequence of the key/-data pair to return.

Returns

The record that is currently in sequence.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 303), **BiometricEvaluation::IO::CompressedRecordStore** (p. 357), **BiometricEvaluation::IO::DBRecordStore** (p. 401), **BiometricEvaluation::IO::FileRecordStore** (p. 443), **BiometricEvaluation::IO::ListRecordStore** (p. 582), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 777).

H.127.3.24 sequenceKey()

```
virtual std::string BiometricEvaluation::IO::RecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT) [pure virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key.

Sequencing means to start at some point in the store and return the key, then repeatedly calling the function to return the next key. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

in	*cursor*	The location within the sequence of the key/-data pair to return.

Returns

The key of the currently sequenced record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 303), **BiometricEvaluation::IO::CompressedRecordStore** (p. 358), **BiometricEvaluation::IO::DBRecordStore** (p. 402), **BiometricEvaluation::IO::FileRecordStore** (p. 444), **BiometricEvaluation::IO::ListRecordStore** (p. 583), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 778).

H.127.3.25 setCursorAtKey()

```
virtual void BiometricEvaluation::IO::RecordStore::setCursorAtKey (
            const std::string & key)  [pure virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 700), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 715).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 715).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 304), **BiometricEvaluation::IO::CompressedRecordStore** (p. 359), **BiometricEvaluation::IO::DBRecordStore** (p. 402), **BiometricEvaluation::IO::FileRecordStore** (p. 444), **BiometricEvaluation::IO::ListRecordStore** (p. 584), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 779).

H.127.3.26 sync()

```
virtual void BiometricEvaluation::IO::RecordStore::sync () const  [pure virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 305), **BiometricEvaluation::IO::CompressedRecordStore** (p. 359), **BiometricEvaluation::IO::DBRecordStore** (p. 403), **BiometricEvaluation::IO::FileRecordStore** (p. 445), **BiometricEvaluation::IO::ListRecordStore** (p. 584), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 779).

H.127.4 Member Data Documentation

H.127.4.1 BE_RECSTORE_SEQ_NEXT

```
const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_NEXT = 2  [static]
```

Tell sequence to sequence from current position

H.127.4.2 BE_RECSTORE_SEQ_START

```
const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_START = 1  [static]
```

Tell **sequence()** (p. 715) to sequence from beginning

H.127.4.3 INVALIDKEYCHARS

```
const std::string BiometricEvaluation::IO::RecordStore::INVALIDKEYCHARS  [static]
```

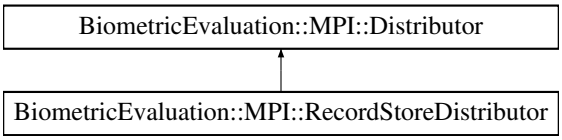
The set of prohibited characters in a key: '/', '\', '*', '&'

H.128 BiometricEvaluation::MPI::RecordStoreDistributor Class Reference

An implementation of the MPI::Distrbutor abstraction that uses a record store for input to create the work packages.

`#include <be_mpi_recordstoredistributor.h>`

Inheritance diagram for BiometricEvaluation::MPI::RecordStoreDistributor:



Public Member Functions

- **RecordStoreDistributor** (const std::string &propertiesFileName, const bool includeValues)

  Construct a distributor using the named properties.

Public Member Functions inherited from BiometricEvaluation::MPI::Distributor

- **Distributor** (const std::string &propertiesFileName)

  Constructor with properties file name.
- void **start** ()

  *Start of **MPI** (p. 162) processing for the distributor.*

Static Public Attributes

- static const std::string **CHECKPOINTLASTKEY**
- static const std::string **CHECKPOINTNUMKEYS**

Static Public Attributes inherited from BiometricEvaluation::MPI::Distributor

- static const std::string **CHECKPOINTFILENAME**
- static const std::string **CHECKPOINTREASON**
- static const std::string **CHECKPOINTPID**

Protected Member Functions

- void **createWorkPackage** (**MPI::WorkPackage** &workPackage)

  Create a work package for distribution.
- void **checkpointSave** (const std::string &reason)

  Create a checkpoint state.
- void **checkpointRestore** ()

  Restore from a checkpoint state.

Protected Member Functions inherited from BiometricEvaluation::MPI::Distributor

- std::shared_ptr< **IO::Logsheet** > **getLogsheet** () const

  Get access to the Logsheet object.
- std::shared_ptr< **IO::PropertiesFile** > **getCheckpointData** () const

  Get access to the checkpoint data object.

H.128.1 Detailed Description

An implementation of the MPI::Distrbutor abstraction that uses a record store for input to create the work packages.

This class supports checkpointing when an early exit is requested, allowing all workers to complete their current work package.

See **MPI::Distributor** (p. 405)

H.128.2 Constructor & Destructor Documentation

H.128.2.1 RecordStoreDistributor()

```
BiometricEvaluation::MPI::RecordStoreDistributor::RecordStoreDistributor (
            const std::string & propertiesFileName,
            const bool includeValues)
```

Construct a distributor using the named properties.

The distributor object is based on the properties given in the file. The name of the input record store must be one of the properties.

The work package sent to Receivers can contain either RecordStore keys, or key/value pairs.

Note

The size of a single value item is limited to 2^32 octets. If the size of the value item is larger, behavior is undefined.

Parameters

in	*propertiesFileName*	The file containing the properties.
in	*includeValues*	true if both the key and value items are included in the work package, false otherwise.

Exceptions

Error::Exception (p. *412*)	An error occurred, typically due to missing or invalid properties.

See also

MPI::Distributor (p. 405)

MPI::RecordProcessor (p. 694)

MPI::RecordStoreResources (p. 727)

H.128.3 Member Function Documentation

H.128.3.1 checkpointRestore()

```
void BiometricEvaluation::MPI::RecordStoreDistributor::checkpointRestore ()  [protected], [virtual]
```

Restore from a checkpoint state.

Implementations of this class use a checkpoint state to move the data sequence cursor to a point past data that has been previously distributed. The **MPI** (p. 162) **Framework** (p. 124) calls this method prior to the start of distributing work packages.

Implements **BiometricEvaluation::MPI::Distributor** (p. 406).

H.128.3.2 checkpointSave()

```
void BiometricEvaluation::MPI::RecordStoreDistributor::checkpointSave (
            const std::string & reason)  [protected], [virtual]
```

Create a checkpoint state.

Implementations of this class create a checkpoint state that captures enough information to allow the implementation to move the data sequence cursor to a point past data that has been previously distributed. The **MPI** (p. 162) **Framework** (p. 124) calls this method when a premature shutdown is requested.

Parameters

reason	A string giving the reason for the checkpoint to be saved.

Implements **BiometricEvaluation::MPI::Distributor** (p. 407).

H.128.3.3 createWorkPackage()

```
void BiometricEvaluation::MPI::RecordStoreDistributor::createWorkPackage (
            MPI::WorkPackage & workPackage)  [protected], [virtual]
```

Create a work package for distribution.

Implementations of this class create a work package to encapsulate the specific data type that is to be distributed.

Implements **BiometricEvaluation::MPI::Distributor** (p. 407).

H.128.4 Member Data Documentation

H.128.4.1 CHECKPOINTLASTKEY

```
const std::string BiometricEvaluation::MPI::RecordStoreDistributor::CHECKPOINTLASTKEY  [static]
```

The last key that was distributed, "Last Key".

H.128.4.2 CHECKPOINTNUMKEYS

```
const std::string BiometricEvaluation::MPI::RecordStoreDistributor::CHECKPOINTNUMKEYS  [static]
```

The number of keys that were distributed, "Num Keys".

H.129 BiometricEvaluation::IO::RecordStoreIterator Class Reference

Generic ForwardIterator for all RecordStores.

`#include <be_io_recordstore.h>`

Public Types

- using **iterator_category** = std::forward_iterator_tag
- using **value_type** = **RecordStore::Record**

- using **difference_type** = std::ptrdiff_t
- using **pointer** = **value_type***
- using **reference** = **value_type**&

Public Member Functions

- **RecordStoreIterator** ()=default

  Default constructor.
- **RecordStoreIterator** (**IO::RecordStore** *recordStore, bool atEnd)

  Constructor.
- **RecordStoreIterator** (const **RecordStoreIterator** &rhs)=default
- **RecordStoreIterator** (**RecordStoreIterator** &&rvalue)=default
- ~**RecordStoreIterator** ()=default
- **reference operator*** ()
- **pointer operator->** ()
- **RecordStoreIterator** & **operator++** ()
- **RecordStoreIterator operator++** (int)
- **RecordStoreIterator operator+=** (**difference_type** rhs)

  Advance a variable number of arguments.
- **RecordStoreIterator operator+** (**difference_type** rhs)

  Advance a variable number of arguments.
- bool **operator==** (const **RecordStoreIterator** &rhs)

  Equivalence operator.
- bool **operator!=** (const **RecordStoreIterator** &rhs)

  Non-equivalence operator.
- **RecordStoreIterator** & **operator=** (const **RecordStoreIterator** &rhs)=default
- **RecordStoreIterator** & **operator=** (**RecordStoreIterator** &&rhs)=default

H.129.1 Detailed Description

Generic ForwardIterator for all RecordStores.

Note

Dereferencing an iterator returns a copy of the value. Modifying a non-const iterator does not manipulate the underlying **RecordStore** (p. 700).

This generic iterator provides no optimization over **RecordStore::sequence()** (p. 715).

H.129.2 Member Typedef Documentation

H.129.2.1 difference_type

```
using BiometricEvaluation::IO::RecordStoreIterator::difference_type = std::ptrdiff_t
```

Type used to measure distance between iterators

H.129.2.2 iterator_category

```
using BiometricEvaluation::IO::RecordStoreIterator::iterator_category = std::forward_iterator_tag
```

Type of iterator

H.129.2.3 pointer

```
using BiometricEvaluation::IO::RecordStoreIterator::pointer = value_type*
```

Pointer to the type iterated over

H.129.2.4 reference

```
using BiometricEvaluation::IO::RecordStoreIterator::reference = value_type&
```

Reference to the type iterated over

H.129.2.5 value_type

```
using BiometricEvaluation::IO::RecordStoreIterator::value_type = RecordStore::Record
```

Type when dereferencing iterators

H.129.3 Constructor & Destructor Documentation

H.129.3.1 RecordStoreIterator() [1/4]

```
BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator () [default]
```

Default constructor.

Creates ”end” iterator.

Note

Satisfies DefaultConstructible requirement.

H.129.3.2 RecordStoreIterator() [2/4]

```
BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator (
            IO::RecordStore * recordStore,
            bool atEnd)
```

Constructor.

Parameters

recordStore	Pointer to a **RecordStore** (p. 700) that will be iterated over.
atEnd	Whether or not to start at the ”end” iterator.

Note

Iterator defaults to starting at the beginning of the **RecordStore** (p. 700).

RecordStoreIterator (p. 721) does not retain any ownership of recordStore.

H.129.3.3 RecordStoreIterator() [3/4]

```
BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator (
            const  RecordStoreIterator & rhs)  [default]
```

Default copy constructor

H.129.3.4 RecordStoreIterator() [4/4]

```
BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator (
             RecordStoreIterator && rvalue)  [default]
```

Default move constructor

H.129.3.5 ~RecordStoreIterator()

```
BiometricEvaluation::IO::RecordStoreIterator::~RecordStoreIterator ()  [default]
```

Default destructor

H.129.4 Member Function Documentation

H.129.4.1 operator"!=()

```
bool BiometricEvaluation::IO::RecordStoreIterator::operator!= (
            const  RecordStoreIterator & rhs)  [inline]
```

Non-equivalence operator.

Parameters

rhs	Reference to **RecordStoreIterator** (p. 721) being compared.

Returns

Whether or not this is not equivalent to rhs.

Note

Satisfies "i != j" is equivalent to "!(i == j)" condition of InputIterator.

H.129.4.2 operator∗()

```
reference BiometricEvaluation::IO::RecordStoreIterator::operator* ()
```

Returns

Reference to a Record.

H.129.4.3 operator+()

```
RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator+ (
            difference_type rhs)
```

Advance a variable number of arguments.

Parameters

rhs	Number of objects to advance (1 or more).

Returns

Self after advancing rhs objects.

H.129.4.4 operator++() [1/2]

```
RecordStoreIterator & BiometricEvaluation::IO::RecordStoreIterator::operator++ ()
```

Returns

Self after advancing.

H.129.4.5 operator++() [2/2]

```
RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator++ (
           int )
```

Returns

Copy of self before advancing.

H.129.4.6 operator+=()

```
RecordStoreIterator BiometricEvaluation::IO::RecordStoreIterator::operator+= (
            difference_type rhs)
```

Advance a variable number of arguments.

Parameters

rhs	Number of objects to advance (1 or more).

Returns

Self after advancing rhs objects.

H.129.4.7 operator->()

```
pointer BiometricEvaluation::IO::RecordStoreIterator::operator-> ()
```

Returns

A dereferenced Record.

H.129.4.8 operator=()

```
RecordStoreIterator & BiometricEvaluation::IO::RecordStoreIterator::operator= (
            RecordStoreIterator && rhs)  [default]
```

Default move assignment operator

H.129.4.9 operator==()

```
bool BiometricEvaluation::IO::RecordStoreIterator::operator== (
            const  RecordStoreIterator & rhs)
```

Equivalence operator.

Parameters

rhs	Reference to **RecordStoreIterator** (p. 721) being compared.

Returns

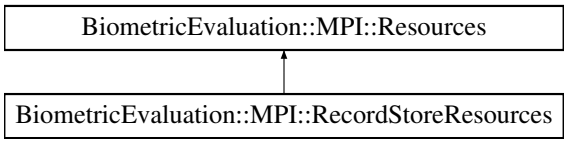
Whether or not this is equivalent to rhs.

H.130 BiometricEvaluation::MPI::RecordStoreResources Class Reference

A class to represent a set of resources needed by an **MPI** (p. 162) program using a RecordStore for input.

`#include <be_mpi_recordstoreresources.h>`

Inheritance diagram for BiometricEvaluation::MPI::RecordStoreResources:



Public Member Functions

- **RecordStoreResources** (const std::string &propertiesFileName)

  Constructor taking the name of the properties file with the resource names.
- uint32_t **getChunkSize** () const
- bool **haveRecordStore** () const

  Indicator that a record store has been opened.
- std::shared_ptr< **IO::RecordStore** > **getRecordStore** () const

  Return the RecordStore named in the property set.

Public Member Functions inherited from BiometricEvaluation::MPI::Resources

- **Resources** (const std::string &propertiesFileName)

  Constructor taking the name of the properties file describing the resources.
- std::string **getPropertiesFileName** () const

  Obtain the name of the file used to construct this object.
- std::string **getLogsheetURL** () const

  *Obtain the Uniform Resource Locator for the **IO** (p. 136):Logsheet object.*
- std::string **getCheckpointPath** () const

  Obtain the Checkpoint Path name.
- int **getRank** () const
- int **getNumTasks** () const
- int **getWorkersPerNode** () const

Static Public Member Functions

- static std::vector< std::string > **getRequiredProperties** ()

  Obtain the required properties as strings.
- static std::vector< std::string > **getOptionalProperties** ()

  Obtain the list of optional properties.

Static Public Member Functions inherited from BiometricEvaluation::MPI::Resources

- static std::vector< std::string > **getRequiredProperties** ()
  Obtain the list of required properties.
- static std::vector< std::string > **getOptionalProperties** ()
  Obtain the list of optional properties.

Static Public Attributes

- static const std::string **INPUTRSPROPERTY**
  The property string ``Input Record Store''; required.
- static const std::string **CHUNKSIZEPROPERTY**
  The property string ``Chunk Size''; required.

Static Public Attributes inherited from BiometricEvaluation::MPI::Resources

- static const std::string **WORKERSPERNODEPROPERTY**
  The property string "Workers Per Node"; required.
- static const std::string **NUMCPUS**
  The "Workers Per Node" setting "NUMCPUS".
- static const std::string **NUMCORES**
  The "Workers Per Node" setting "NUMCORES".
- static const std::string **NUMSOCKETS**
  The "Workers Per Node" setting "NUMSOCKETS".
- static const std::string **LOGSHEETURLPROPERTY**
  The property string "Logsheet URL"; optional.
- static const std::string **CHECKPOINTPATHPROPERTY**
  The property string "Checkpoint Path"; required when checkpointing is enabled, optional otherwise.

H.130.1 Detailed Description

A class to represent a set of resources needed by an **MPI** (p. 162) program using a RecordStore for input.

Resources (p. 740) are opened based on the property when appropriate. The input record store need not be accessible. Applications should call **haveRecordStore()** (p. 729) to check whether the record store has been opened.

H.130.2 Constructor & Destructor Documentation

H.130.2.1 RecordStoreResources()

```
BiometricEvaluation::MPI::RecordStoreResources::RecordStoreResources (
            const std::string & propertiesFileName)
```

Constructor taking the name of the properties file with the resource names.

Exceptions

Error::FileError *(p. 420)*	The resources file could not be read.
Error::ObjectDoesNotExist *(p. 637)*	A required property does not exist.
Error::Exception *(p. 412)*	Some other error occurred.

H.130.3 Member Function Documentation

H.130.3.1 getOptionalProperties()

```
static std::vector< std::string > BiometricEvaluation::MPI::RecordStoreResources::getOptional↩
Properties () [static]
```

Obtain the list of optional properties.

Returns

A set of optional property strings.

H.130.3.2 getRecordStore()

```
std::shared_ptr< IO::RecordStore > BiometricEvaluation::MPI::RecordStoreResources::getRecord↩
Store () const
```

Return the RecordStore named in the property set.

Returns

A shared pointer to the record store.

H.130.3.3 getRequiredProperties()

```
static std::vector< std::string > BiometricEvaluation::MPI::RecordStoreResources::getRequired↩
Properties () [static]
```

Obtain the required properties as strings.

Returns

The set of required properties.

H.130.3.4 haveRecordStore()

```
bool BiometricEvaluation::MPI::RecordStoreResources::haveRecordStore () const
```

Indicator that a record store has been opened.

Returns

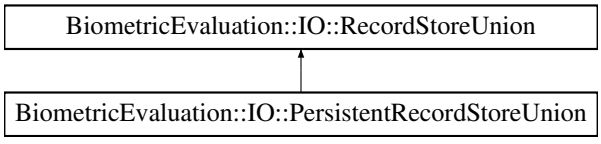
true if input record store is opened, false otherwise.

H.131 BiometricEvaluation::IO::RecordStoreUnion Class Reference

A collection of N related read-only RecordStores, operated on simultaneously.

`#include <be_io_recordstoreunion.h>`

Inheritance diagram for BiometricEvaluation::IO::RecordStoreUnion:



Public Member Functions

- **RecordStoreUnion** (const std::map< const std::string, const std::string > &recordStores)
- **RecordStoreUnion** (std::map< const std::string, const std::string >::iterator first, std::map< const std::string, const std::string >::iterator last)
- **RecordStoreUnion** (std::initializer_list< std::pair< const std::string, const std::string > > recordStores)
- **RecordStoreUnion** (const std::map< const std::string, const std::shared_ptr< **BiometricEvaluation::IO::RecordStore** > > &recordStores)
- **RecordStoreUnion** (std::map< const std::string, const std::shared_ptr< **BiometricEvaluation::IO::RecordStore** > >::iterator first, std::map< const std::string, const std::shared_ptr< **BiometricEvaluation::IO::RecordStore** > >::iterator last)
- **RecordStoreUnion** (std::initializer_list< std::pair< const std::string, const std::shared_ptr< **BiometricEvaluation::IO::RecordStore** > > > recordStores)
- std::shared_ptr< **BiometricEvaluation::IO::RecordStore** > **getRecordStore** (const std::string &name) const

  *Obtain a pointer to an open **RecordStore*** (p. 700).
- std::vector< std::string > **getNames** () const

  Obtain the names of RecordStores set during construction.
- std::map< const std::string, **BiometricEvaluation::Memory::uint8Array** > **read** (const std::string &key) const

  Read a key from all member RecordStores.
- std::map< const std::string, uint64_t > **length** (const std::string &key) const

  Retrieve the length of a key from all member RecordStores.
- **RecordStoreUnion** (const **RecordStoreUnion** &)=delete
- **RecordStoreUnion** & **operator=** (const **RecordStoreUnion** &)=delete
- ~**RecordStoreUnion** ()

Protected Member Functions

- **RecordStoreUnion** ()

  Empty constructor for children.
- void **setImpl** (const std::shared_ptr< RecordStoreUnion::Impl > &pimpl)

  Change the implementation of this object.

H.131.1 Detailed Description

A collection of N related read-only RecordStores, operated on simultaneously.

A **RecordStoreUnion** (p. 729) object is not copyable due to the fact that most **RecordStore** (p. 700) objects are not copyable.

H.131.2 Constructor & Destructor Documentation

H.131.2.1 RecordStoreUnion() [1/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion (
            const std::map< const std::string, const std::string > & recordStores)
```

RecordStoreUnion (p. 729) constructor.

Parameters

recordStores	Map of developer-provided names to paths to a **RecordStore** (p. 700).

H.131.2.2 RecordStoreUnion() [2/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion (
            std::map< const std::string, const std::string >::iterator first,
            std::map< const std::string, const std::string >::iterator last)
```

RecordStoreUnion (p. 729) constructor.

Parameters

first	Iterator to the start of a map of developer-provided names to paths to a **RecordStore** (p. 700).
last	Iterator to the end of a map of developer-provided names to paths to a **RecordStore** (p. 700).

H.131.2.3 RecordStoreUnion() [3/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion (
            std::initializer_list< std::pair< const std::string, const std::string > > recordStores)
```

RecordStoreUnion (p. 729) constructor.

Parameters

recordStores	List of pairs of developer-provided name and paths to a **RecordStore** (p. 700).

H.131.2.4 RecordStoreUnion() [4/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion (
            const std::map< const std::string, const std::shared_ptr< BiometricEvaluation::IO::RecordStore > > & recordStores)
```

RecordStoreUnion (p. 729) constructor.

Parameters

recordStores	Map of developer-provided names and open **RecordStore** (p. 700) objects.

Note

Behavior when providing a **RecordStore** (p. 700) that has been opened read/write is undefined.

H.131.2.5 RecordStoreUnion() [5/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion (
            std::map< const std::string, const std::shared_ptr< BiometricEvaluation::IO::↩
RecordStore > >::iterator first,
            std::map< const std::string, const std::shared_ptr< BiometricEvaluation::IO::↩
RecordStore > >::iterator last)
```

RecordStoreUnion (p. 729) constructor.

Parameters

first	Iterator to the start of a map of developer-provided names and open **RecordStore** (p. 700) objects.
last	Iterator to the end of a map of developer-provided names and open **RecordStore** (p. 700) objects.

Note

Behavior when providing a **RecordStore** (p. 700) that has been opened read/write is undefined.

H.131.2.6 RecordStoreUnion() [6/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion (
            std::initializer_list< std::pair< const std::string, const std::shared_ptr< Biometric↩
Evaluation::IO::RecordStore > > > recordStores)
```

RecordStoreUnion (p. 729) constructor.

Parameters

recordStores	List of pairs of developer-provided name and open **RecordStore** (p. 700) objects.

Note

Behavior when providing a **RecordStore** (p. 700) that has been opened read/write is undefined.

H.131.2.7 ~RecordStoreUnion()

`BiometricEvaluation::IO::RecordStoreUnion::~RecordStoreUnion ()`

Destructor.

H.131.2.8 RecordStoreUnion() [7/7]

`BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion () [protected]`

Empty constructor for children.

Note

Implementation is not set. Callers must also call **setImpl()** (p. 737) to provide functionality.

@seealso setImpl

H.131.3 Member Function Documentation

H.131.3.1 getNames()

`std::vector< std::string > BiometricEvaluation::IO::RecordStoreUnion::getNames () const`

Obtain the names of RecordStores set during construction.

Returns

Vector of names of RecordStores.

H.131.3.2 getRecordStore()

`std::shared_ptr< BiometricEvaluation::IO::RecordStore > BiometricEvaluation::IO::RecordStoreUnion::getRecordStore (`
`const std::string & name) const`

Obtain a pointer to an open **RecordStore** (p. 700).

Parameters

name	Name provided to **RecordStore** (p. 700) during construction.

Exceptions

ObjectDoesNotExist	name is not recognized.

H.131.3.3 length()

```
std::map< const std::string, uint64_t > BiometricEvaluation::IO::RecordStoreUnion::length (
            const std::string & key) const
```

Retrieve the length of a key from all member RecordStores.

Parameters

key	The key to read.

Returns

Map of **RecordStore** (p. 700) name to data length read from said **RecordStore** (p. 700).

Exceptions

Error::ObjectDoesNotExist (p. 637)	key does not exist in any member RecordStores.
Error::StrategyError (p. 789)	Exceptions propagated from **RecordStore** (p. 700), with the exception of ObjectDoesNotE

Note

Exceptions are thrown after **length()** (p. 736) has been called on all member RecordStores.

H.131.3.4 read()

```
std::map< const std::string, BiometricEvaluation::Memory::uint8Array > BiometricEvaluation
::IO::RecordStoreUnion::read (
            const std::string & key) const
```

Read a key from all member RecordStores.

Parameters

key	The key to read.

Returns

Map of **RecordStore** (p. 700) name to data read from said **RecordStore** (p. 700).

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	key does not exist in any member RecordStores.
Error::StrategyError *(p. 789)*	Exceptions propagated from **RecordStore** (p. 700), with the exception of ObjectDoesNotE

Note

Exceptions are thrown after **read()** (p. 736) has been called on all member RecordStores.

H.131.3.5 setImpl()

```
void BiometricEvaluation::IO::RecordStoreUnion::setImpl (
            const std::shared_ptr< RecordStoreUnion::Impl > & pimpl)  [protected]
```

Change the implementation of this object.

Parameters

impl	Pointer to an implementation instance.

H.132 BiometricEvaluation::Image::Resolution Struct Reference

A structure to represent the resolution of an image.

`#include <be_image.h>`

Public Types

- enum class **Units** { **NA** = 0 , **PPI** = 1 , **PPMM** = 2 , **PPCM** = 3 }

  *Possible representations of the units in a **Resolution** (p. 737) struct.*

Public Member Functions

- **Resolution** (const double **xRes**=0.0, const double **yRes**=0.0, const **Units** **units**= **Units::PPI**)

  *Create a **Resolution** (p. 737) struct.*
- **Resolution** **toUnits** (const **Units** & **units**) const

  Obtain alternate representations of this resolution.

Public Attributes

- double **xRes**
- double **yRes**
- **Units** **units**

H.132.1 Detailed Description

A structure to represent the resolution of an image.

H.132.2 Member Enumeration Documentation

H.132.2.1 Units

`enum class` **`BiometricEvaluation::Image::Resolution::Units`** `[strong]`

Possible representations of the units in a **Resolution** (p. 737) struct.

Enumerator

NA	Not-applicable: unknown, or otherwise
PPI	Pixels per inch
PPMM	Pixels per millimeter
PPCM	Pixels per centimeter

H.132.3 Constructor & Destructor Documentation

H.132.3.1 Resolution()

```
BiometricEvaluation::Image::Resolution::Resolution (
            const double xRes = 0.0,
```

```
const double yRes = 0.0,
const  Units units =  Units::PPI)
```

Create a **Resolution** (p. 737) struct.

Parameters

in	*xRes*	**Resolution** (p. 737) along the X-axis
in	*yRes*	**Resolution** (p. 737) along the Y-axis
in	*units*	Units in which xRes and yRes are represented

H.132.4 Member Function Documentation

H.132.4.1 toUnits()

```
Resolution BiometricEvaluation::Image::Resolution::toUnits (
            const  Units & units) const
```

Obtain alternate representations of this resolution.

Parameters

units	The units to which this resolution is converted.

Returns

This resolution, in units units.

Exceptions

BE::Error::StrategyError	Units are not defined for either the source or destination resolution.

H.132.5 Member Data Documentation

H.132.5.1 units

`Units BiometricEvaluation::Image::Resolution::units`

Units in which xRes and yRes are represented

H.132.5.2 xRes

`double BiometricEvaluation::Image::Resolution::xRes`

Resolution (p. 737) along the X-axis

H.132.5.3 yRes

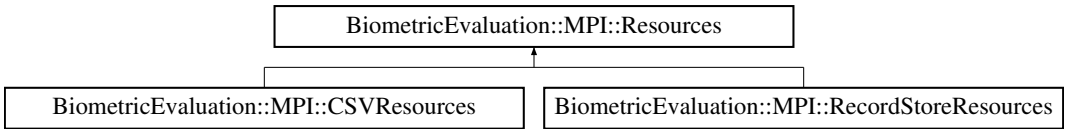
`double BiometricEvaluation::Image::Resolution::yRes`

Resolution (p. 737) along the Y-axis

H.133 BiometricEvaluation::MPI::Resources Class Reference

`#include <be_mpi_resources.h>`

Inheritance diagram for BiometricEvaluation::MPI::Resources:



Public Member Functions

- **Resources** (const std::string &propertiesFileName)

  Constructor taking the name of the properties file describing the resources.
- std::string **getPropertiesFileName** () const

  Obtain the name of the file used to construct this object.
- std::string **getLogsheetURL** () const

  *Obtain the Uniform Resource Locator for the **IO** (p. 136):Logsheet object.*
- std::string **getCheckpointPath** () const

  Obtain the Checkpoint Path name.
- int **getRank** () const
- int **getNumTasks** () const
- int **getWorkersPerNode** () const

Static Public Member Functions

- static std::vector< std::string > **getRequiredProperties** ()

  Obtain the list of required properties.
- static std::vector< std::string > **getOptionalProperties** ()

  Obtain the list of optional properties.

Static Public Attributes

- static const std::string **WORKERSPERNODEPROPERTY**

  The property string "Workers Per Node"; required.
- static const std::string **NUMCPUS**

  The "Workers Per Node" setting "NUMCPUS".
- static const std::string **NUMCORES**

  The "Workers Per Node" setting "NUMCORES".
- static const std::string **NUMSOCKETS**

  The "Workers Per Node" setting "NUMSOCKETS".
- static const std::string **LOGSHEETURLPROPERTY**

  The property string "Logsheet URL"; optional.
- static const std::string **CHECKPOINTPATHPROPERTY**

  The property string "Checkpoint Path"; required when checkpointing is enabled, optional otherwise.

H.133.1 Detailed Description

A class to represent a set of resources needed by an **MPI** (p. 162) program. The resources are based on a properties file as well as some dynamic information, such as **MPI** (p. 162) rank and process ID.

H.133.2 Constructor & Destructor Documentation

H.133.2.1 Resources()

```
BiometricEvaluation::MPI::Resources::Resources (
            const std::string & propertiesFileName)
```

Constructor taking the name of the properties file describing the resources.

Parameters

in	*propertiesFileName*	The name of the file containing the Properties.

Exceptions

Error::FileError *(p. 420)*	The resources file could not be read.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A required property does not exist.
Error::Exception (p. 412)	Some other error occurred.

H.133.3 Member Function Documentation

H.133.3.1 getCheckpointPath()

```
std::string BiometricEvaluation::MPI::Resources::getCheckpointPath () const
```

Obtain the Checkpoint Path name.

This string my be empty, indicating that there is no checkpoint path in the Properties file.

Returns

The Checkpoint Path.

H.133.3.2 getLogsheetURL()

```
std::string BiometricEvaluation::MPI::Resources::getLogsheetURL () const
```

Obtain the Uniform Resource Locator for the **IO** (p. 136):Logsheet object.

This string my be empty, indicating that there is no Logsheet URL in the Properties file.

Returns

The Logsheet URL.

H.133.3.3 getOptionalProperties()

```
static std::vector< std::string > BiometricEvaluation::MPI::Resources::getOptionalProperties
() [static]
```

Obtain the list of optional properties.

Returns

A set of optional property strings.

H.133.3.4 getPropertiesFileName()

```
std::string BiometricEvaluation::MPI::Resources::getPropertiesFileName () const
```

Obtain the name of the file used to construct this object.

Returns

The name of the properties file.

H.133.3.5 getRequiredProperties()

```
static std::vector< std::string > BiometricEvaluation::MPI::Resources::getRequiredProperties
() [static]
```

Obtain the list of required properties.

Returns

A set of required property strings.

H.133.4 Member Data Documentation

H.133.4.1 NUMCORES

```
const std::string BiometricEvaluation::MPI::Resources::NUMCORES  [static]
```

The "Workers Per Node" setting "NUMCORES".

This setting indicates the **MPI** (p. 162) **Framework** (p. 124) is to create one worker for each physical CPU core.

H.133.4.2 NUMCPUS

```
const std::string BiometricEvaluation::MPI::Resources::NUMCPUS  [static]
```

The "Workers Per Node" setting "NUMCPUS".

This setting indicates the **MPI** (p. 162) **Framework** (p. 124) is to create one worker for each logical CPU.

H.133.4.3 NUMSOCKETS

```
const std::string BiometricEvaluation::MPI::Resources::NUMSOCKETS  [static]
```

The "Workers Per Node" setting "NUMSOCKETS".

This setting indicates the **MPI** (p. 162) **Framework** (p. 124) is to create one worker for each physical CPU socket.

H.133.4.4 WORKERSPERNODEPROPERTY

```
const std::string BiometricEvaluation::MPI::Resources::WORKERSPERNODEPROPERTY  [static]
```

The property string "Workers Per Node"; required.

This value shall be either an integer or one of the strings "NUMCPUS", "NUMCORES", "NUMSOCKETS".

H.134 BiometricEvaluation::Framework::API< T >::Result Class Reference

```
#include <be_framework_api.h>
```

Public Member Functions

- **Result** ()
- bool **operator!** () const

  Logical negation operator overload.
- **operator bool** () const

  Boolean conversion operator.
- std::string **getExceptionStr** () const noexcept

  Obtain the exception string.
- void **rethrowException** () const

  Rethrow the caught exception.
- void **setException** (std::exception_ptr e)

  Save a thrown exception.
- template<typename Duration >
  std::uintmax_t **elapsed** () const

Public Attributes

- std::common_type_t< Time::Timer::BE_CLOCK_TYPE::time_point::duration, Time::Timer::BE_CLOCK↩ _TYPE::time_point::duration > **elapsedTimePoint**
- T **status**

  Value returned from operation.
- **APICurrentState currentState**

  Current state of operation.

H.134.1 Detailed Description

template<typename T>
class BiometricEvaluation::Framework::API< T >::Result

The result of an operation.

H.134.2 Constructor & Destructor Documentation

H.134.2.1 Result()

```
template<typename T >
BiometricEvaluation::Framework::API< T >::Result::Result ()
```

Constructor

H.134.3 Member Function Documentation

H.134.3.1 elapsed()

```
template<typename T >
template<typename Duration >
std::uintmax_t  BiometricEvaluation::Framework::API< T >::Result::elapsed () const  [inline]
```

Returns

Integral value representing elapsed time.

H.134.3.2 getExceptionStr()

```
template<typename T >
std::string  BiometricEvaluation::Framework::API< T >::Result::getExceptionStr () const  [inline],
[noexcept]
```

Obtain the exception string.

Returns

Explanatory message of the exception thrown if the exception is derived from std::exception, or a default-initialized string otherwise.

H.134.3.3 operator bool()

```
template<typename T >
BiometricEvaluation::Framework::API< T >::Result::operator bool () const  [inline], [explicit]
```

Boolean conversion operator.

Returns

True if operation completed, false otherwise.

H.134.3.4 operator"!()

```
template<typename T >
bool  BiometricEvaluation::Framework::API< T >::Result::operator!  () const  [inline]
```

Logical negation operator overload.

Returns

True if operation failed to complete, false otherwise.

H.134.3.5 rethrowException()

```
template<typename T >
void  BiometricEvaluation::Framework::API< T >::Result::rethrowException () const  [inline]
```

Rethrow the caught exception.

This is useful for applications by allowing them to examine an exception thrown during **call()** (p. 287) from either success or failure callback when rethrowExceptions is false. It also prevents needing to define a verbose type outside a try/catch block when rethrowExceptions is true.

Note

If no exception was caught, an exception will still be thrown.

Exceptions

Always	throws.

H.134.3.6 setException()

```
template<typename T >
void  BiometricEvaluation::Framework::API< T >::Result::setException (
            std::exception_ptr e)  [inline]
```

Save a thrown exception.

Parameters

e	Pointer to exception caught.

H.134.4 Member Data Documentation

H.134.4.1 elapsedTimePoint

```
template<typename T >
std::common_type_t< Time::Timer::BE_CLOCK_TYPE::time_point::  duration, Time::Timer::BE_CLOCK↩
_TYPE::time_point::  duration>  BiometricEvaluation::Framework::API< T >::Result::elapsedTime↩
Point
```

Time (p. 185) elapsed while calling operation.

H.134.4.2 status

```
template<typename T >
T  BiometricEvaluation::Framework::API< T >::Result::status
```

Value returned from operation.

Note

Only populated when currentState == **APICurrentState::Completed** (p. **??**).

H.135 BiometricEvaluation::Feature::RidgeCountItem Struct Reference

Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number.

`#include <be_feature_minutiae.h>`

Public Member Functions

- **RidgeCountItem** (**RidgeCountExtractionMethod** extraction_method, int index_one, int index_two, int count=0)

  *Create a **RidgeCountItem** (p. 746) struct.*

Public Attributes

- **RidgeCountExtractionMethod extraction_method**
- int **index_one**
- int **index_two**
- int **count**

H.135.1 Detailed Description

Representation of ridge count data, which is the number of ridges between any two minutia data points, each represented by its index number.

H.136 BiometricEvaluation::Image::ROI Struct Reference

A structure to represent a region of interest (**ROI** (p. 746)), which is a bounding box and a set of coordinates.

`#include <be_image.h>`

Public Member Functions

- **ROI** ()
- **ROI** (const **Size** size, const uint32_t horzOffset, const uint32_t vertOffset, const CoordinateSet &path)

Public Attributes

- **Size size**
- uint32_t **horzOffset**
- uint32_t **vertOffset**
- CoordinateSet **path**

H.136.1 Detailed Description

A structure to represent a region of interest (**ROI** (p. 746)), which is a bounding box and a set of coordinates.

H.136.2 Constructor & Destructor Documentation

H.136.2.1 ROI() [1/2]

```
BiometricEvaluation::Image::ROI::ROI ()
```

Create an empty **ROI** (p. 746) object.

H.136.2.2 ROI() [2/2]

```
BiometricEvaluation::Image::ROI::ROI (
            const  Size size,
            const uint32_t horzOffset,
            const uint32_t vertOffset,
            const CoordinateSet & path)
```

Create a **ROI** (p. 746) object with the given parameters.

Parameters

in	*size*	The size of the region of interest.
in	*horzOffset*	The horizontal offset of the region of interest.

Parameters

in	*vertOffset*	The vertical offset of the region of interest.
in	*path*	The path offset of the region of interest.

H.137 BiometricEvaluation::MPI::Runtime Class Reference

Runtime (p. 748) support for the startup/shutdown of **MPI** (p. 162) jobs.

`#include <be_mpi_runtime.h>`

Public Member Functions

- **Runtime** (int &argc, char **&argv, bool checkpointEnable=false)

  *Construct the runtime environment for the processes making up the **MPI** (p. 162) job.*
- void **start** (**BiometricEvaluation::MPI::Distributor** &distributor, **BiometricEvaluation::MPI::Receiver** &receiver)

  *Startup the runtime environment for the **MPI** (p. 162) job.*
- void **shutdown** ()

  *Shutdown the runtime environment for the **MPI** (p. 162) job.*
- void **abort** (int errcode)

  *Abort the runtime the **MPI** (p. 162) job.*

H.137.1 Detailed Description

Runtime (p. 748) support for the startup/shutdown of **MPI** (p. 162) jobs.

This class provides methods that are used by applications to start and shutdown the **MPI** (p. 162) job. Each job consists of a single distributor of work, and 1..n receivers of work which then distribute the work packages to child processes to take action on the work package.

H.137.2 Constructor & Destructor Documentation

H.137.2.1 Runtime()

```
BiometricEvaluation::MPI::Runtime::Runtime (
            int & argc,
```

```
        char **& argv,
        bool checkpointEnable = false)
```

Construct the runtime environment for the processes making up the **MPI** (p. 162) job.

Whether to save a checkpoint on clean shutdown, and recover a checkpoint on startup, is optionally specified.

Parameters

in	*argc*	The argument count, taken from the command line passed to main().
in	*argv*	The argument vector, taken from the command line passed to main().

Parameters

in	*checkpointEnable*	True indicates that a checkpoint should be saved on early shutdown and restored on startup, if the checkpoint data is present.. Checkpoints are implementation-defined by the **Distributor** (p. 405) classes.

H.137.3 Member Function Documentation

H.137.3.1 abort()

```
void BiometricEvaluation::MPI::Runtime::abort (
            int errcode)
```

Abort the runtime the **MPI** (p. 162) job.

This method will cause the **MPI** (p. 162) job to terminate immediately. All processes will end without the opportunity to save.

Parameters

in	*errocode*	The error code to return to the **MPI** (p. 162) runtime.

H.137.3.2 shutdown()

```
void BiometricEvaluation::MPI::Runtime::shutdown ()
```

Shutdown the runtime environment for the **MPI** (p. 162) job.

This method must be called in order for the **MPI** (p. 162) runtime to cleanly exit.

H.137.3.3 start()

```
void BiometricEvaluation::MPI::Runtime::start (
            BiometricEvaluation::MPI::Distributor & distributor,
            BiometricEvaluation::MPI::Receiver & receiver)
```

Startup the runtime environment for the **MPI** (p. 162) job.

Exceptions thrown by the **Distributor** (p. 405) or Recevier are caught and logged.

Parameters

in	*distributor*	The **Distributor** (p. 405) object that will form the basis of the first **MPI** (p. 162) task.

Parameters

in	*receiver*	The **Receiver** (p. 692) object which will form the basis of **MPI** (p. 162) tasks 1..n.

H.138 BiometricEvaluation::Process::Semaphore Class Reference

Represent a semaphore that can be used for interprocess communication.

`#include <be_process_semaphore.h>`

Public Member Functions

- **Semaphore** (const std::string &name, const mode_t mode, const int value, const bool force=false)

  Create a new named sempahore.
- **Semaphore** (const std::string &name)

  Open an existing named sempahore.
- bool **wait** (const bool interruptible)

  Wait indefinitely for the semaphore to unblock.
- bool **trywait** (const bool interruptible)

  Attempt to obtain the semaphore without blocking.
- bool **timedwait** (const uint64_t interval, const bool interruptible)

  Attempt to obtain the semaphore while blocking for at most the specified time interval.
- void **post** ()

  Post (increment) to the semaphore.
- std::string **getName** ()

  *Obtain the name of the **Semaphore** (p. 752).*

H.138.1 Detailed Description

Represent a semaphore that can be used for interprocess communication.

Semaphores are shared counters with mutually exclusive modification properties. A counter value greater than zero means that a resource represented by the semaphore is available. A typical use is to grant exclusive access to a resource by allowing the counter to be valued at zero or one; this is known as a binary semaphore.

Note

The counter value is not exposed to clients of the object.

Because a **Semaphore** (p. 752) object wraps a system resource, the **Semaphore** (p. 752) can be passed to other functions, or inherited across a fork boundary.

H.138.2 Constructor & Destructor Documentation

H.138.2.1 Semaphore() [1/2]

```
BiometricEvaluation::Process::Semaphore::Semaphore (
            const std::string & name,
            const mode_t mode,
            const int value,
            const bool force = false)
```

Create a new named sempahore.

Parameters

in	*name*	The name of the semaphore, which must obey the syntax documented for the sem↩ _↩ open(2) call. If the semaphore already exists in the name space, construction will fail unless the force flag is true. In that case, the existing semaphore will be removed.
in	*mode*	The permission mode of the semaphore.

Parameters

in	*value*	The initial value of the semaphore.
in	*force*	The semaphore is created, disassociating an existing semaphore of the same name.

Exceptions

Error::ObjectExists (p. 637)	The semaphore already exists with the given name.
Error::StrategyError (p. 789)	An error occurred when creating the semaphore.

H.138.2.2 Semaphore() [2/2]

```
BiometricEvaluation::Process::Semaphore::Semaphore (
            const std::string & name)
```

Open an existing named sempahore.

Parameters

in	*name*	The name of the semaphore, which must obey the syntax documented for the sem↩ _↩ open(2) call.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	A semaphore does not exist with the given name.
Error::StrategyError *(p. 789)*	An error occurred when creating the semaphore.

H.138.3 Member Function Documentation

H.138.3.1 getName()

```
std::string BiometricEvaluation::Process::Semaphore::getName ()
```

Obtain the name of the **Semaphore** (p. 752).

Returns

The name of the Sempahore.

H.138.3.2 post()

```
void BiometricEvaluation::Process::Semaphore::post ()
```

Post (increment) to the semaphore.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	The semaphore is no longer valid.
Error::StrategyError *(p. 789)*	**System** (p. 171) error obtaining the semaphore.

H.138.3.3 timedwait()

```
bool BiometricEvaluation::Process::Semaphore::timedwait (
            const uint64_t interval,
            const bool interruptible)
```

Attempt to obtain the semaphore while blocking for at most the specified time interval.

Parameters

in	*interval*	The max time to wait, in microseconds.

Parameters

in	*interruptible*	true if the function should return if waiting was interrupted, false otherwise.

Returns

true if the semaphore was obtained; false if not.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The semaphore is no longer valid.
Error::NotImplemented (p. 636)	Function is not implemented on the system. Applications should then call **wait()** (p. 758) o
Error::StrategyError (p. 789)	**System** (p. 171) error obtaining the semaphore.

H.138.3.4 trywait()

```
bool BiometricEvaluation::Process::Semaphore::trywait (
            const bool interruptible)
```

Attempt to obtain the semaphore without blocking.

Parameters

in	*interruptible*	true if the function should return if waiting was interrupted, false otherwise.

Returns

true if the semaphore was obtained; false if not.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The semaphore is no longer valid.
Error::StrategyError (p. 789)	**System** (p. 171) error obtaining the semaphore.

H.138.3.5 wait()

```
bool BiometricEvaluation::Process::Semaphore::wait (
            const bool interruptible)
```

Wait indefinitely for the semaphore to unblock.

Parameters

in	*interruptible*	true if the function should return if waiting was interrupted, false otherwise.

Returns

true if the semaphore was obtained; false if not.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	The semaphore is no longer valid.
Error::StrategyError *(p. 789)*	**System** (p. 171) error obtaining the semaphore.

H.139 BiometricEvaluation::Error::SignalManager Class Reference

A **SignalManager** (p. 759) object is used to handle signals that come from the operating system.

`#include <be_error_signal_manager.h>`

Public Member Functions

- **SignalManager** ()
- **SignalManager** (const sigset_t signalSet)
- void **setSignalSet** (const sigset_t signalSet)
- void **clearSignalSet** ()
- void **setDefaultSignalSet** ()
- bool **sigHandled** ()
- void **start** ()
- void **stop** ()
- void **setSigHandled** ()
- void **clearSigHandled** ()
- void **setEnabled** (const bool enabled)
- bool **isEnabled** () const

Static Public Attributes

- static bool **_canSigJump**
- static sigjmp_buf **_sigJumpBuf**

H.139.1 Detailed Description

A **SignalManager** (p. 759) object is used to handle signals that come from the operating system.

Applications typically do not invoke most methods of a **SignalManager** (p. 759), except the **setSignal↩Set()** (p. 761), **setDefaultSignalSet()** (p. 761), and **sigHandled()** (p. 762). An application wishing to just catch memory errors can simply construct a **SignalManager** (p. 759) object, and invoke **sigHandled()** (p. 762) 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** (p. 759) object to start handling signals. Applications can call either **setSignalSet()** (p. 761) or **setDefaultSignalSet()** (p. 761) before invoking these macros to indicate which signals are to be handled.

The END_SIGNAL_BLOCK() macro clears the signal set, so from that point forward application code signals will be handled in the system's default manner until another signal block is created.

The ABORT_SIGNAL_MANAGER() macro also disables the watchdog timer but does not create the code point destination for the jump point. This macro should be used to disable a **SignalManager** (p. 759) object when the application is no longer interested in the signal handling.

Attention

The BEGIN_SIGNAL_BLOCK() macro must be paired with either the END_SIGNAL_BLOCK() macro or ABORT_SIGNAL_MANAGER() macro. Failure to do so may result in undefined behavior as an active **SignalManager** (p. 759) may be invoked, forcing a jump into an incompletely initialized function.

A **SignalManager** (p. 759) is passive (i.e. no signal handlers are installed) until that **start()** (p. 762) method is called, and becomes passive when **stop()** (p. 762) 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()** (p. 762) is called.

Attention

The **start()** (p. 762), **stop()** (p. 762), **setSigHandled()** (p. 761) and **clearSigHandled()** (p. 761) methods are not meant to be used directly by applications, which should use the BEGIN_SIGNAL_BLOCK()/↩ END_SIGNAL_BLOCK() macro pair.

H.139.2 Constructor & Destructor Documentation

H.139.2.1 SignalManager() [1/2]

```
BiometricEvaluation::Error::SignalManager::SignalManager ()
```

Construct a new **SignalManager** (p. 759) object with the default signal handling: SIGSEGV and SIGBUS.

Exceptions

Error::StrategyError *(p. 789)*	Could not register the signal handler.

H.139.2.2 SignalManager() [2/2]

```
BiometricEvaluation::Error::SignalManager::SignalManager (
            const sigset_t signalSet)
```

Construct a new **SignalManager** (p. 759) 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 *(p. 655)*	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

H.139.3 Member Function Documentation

H.139.3.1 clearSigHandled()

```
void BiometricEvaluation::Error::SignalManager::clearSigHandled ()
```

Clear the indication that a signal was handled.

H.139.3.2 clearSignalSet()

```
void BiometricEvaluation::Error::SignalManager::clearSignalSet ()
```

Clear all signal handling.

H.139.3.3 isEnabled()

```
bool BiometricEvaluation::Error::SignalManager::isEnabled () const
```

Check the enabled status of signal handling.

H.139.3.4 setDefaultSignalSet()

```
void BiometricEvaluation::Error::SignalManager::setDefaultSignalSet ()
```

Set the default signals this object will manage: SIGSEGV and SIGBUS.

H.139.3.5 setEnabled()

```
void BiometricEvaluation::Error::SignalManager::setEnabled (
            const bool enabled)
```

Enable or disable signal handling.

Parameters

enabled	`true` if enabled, `false` otherwise.

Note

This enables easier debugging without changing sourcecode to remove **SignalManager** (p. 759) blocks.

H.139.3.6 setSigHandled()

```
void BiometricEvaluation::Error::SignalManager::setSigHandled ()
```

Set a flag to indicate a signal was handled.

H.139.3.7 setSignalSet()

```
void BiometricEvaluation::Error::SignalManager::setSignalSet (
            const sigset_t signalSet)
```

Set the signals this object will manage.

Parameters

signalSet	(in) The signal set; see sigaction(2), sigemptyset(3) and sigaddset(3).

Exceptions

Error::ParameterError *(p. 655)*	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

H.139.3.8 sigHandled()

`bool BiometricEvaluation::Error::SignalManager::sigHandled ()`

Indicate whether a signal was handled.

Returns

true if a signal was handled, false otherwise.

H.139.3.9 start()

`void BiometricEvaluation::Error::SignalManager::start ()`

Start handling signals of the current signal set.

Exceptions

Error::StrategyError *(p. 789)*	Could not register the signal handler.

Note

If an application invokes **start()** (p. 762) 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.

H.139.3.10 stop()

`void BiometricEvaluation::Error::SignalManager::stop ()`

Stop handling signals of the current signal set.

Exceptions

Error::StrategyError *(p. 789)*	Could not register the signal handler.

H.139.4 Member Data Documentation

H.139.4.1 _canSigJump

```
bool BiometricEvaluation::Error::SignalManager::_canSigJump  [static]
```

Flag indicating can jump after handling a signal.

Note

Should not be directly used by applications.

H.139.4.2 _sigJumpBuf

```
sigjmp_buf BiometricEvaluation::Error::SignalManager::_sigJumpBuf  [static]
```

The jump buffer used by the signal handler.

Note

Should not be directly used by applications.

H.140 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** (p. 763) struct.*

Public Attributes

- uint32_t **xSize**
- uint32_t **ySize**

H.140.1 Detailed Description

A structure to represent the size of an image, in pixels.

H.140.2 Constructor & Destructor Documentation

H.140.2.1 Size()

```
BiometricEvaluation::Image::Size::Size (
            const uint32_t xSize = 0,
            const uint32_t ySize = 0)
```

Create a **Size** (p. 763) struct.

Parameters

in	*xSize*	Number of pixels on the X-axis
in	*ySize*	Number of pixels on the Y-axis

H.140.3 Member Data Documentation

H.140.3.1 xSize

`uint32_t BiometricEvaluation::Image::Size::xSize`

Number of pixels on the X-axis

H.140.3.2 ySize

`uint32_t BiometricEvaluation::Image::Size::ySize`

Number of pixels on the Y-axis

H.141 BiometricEvaluation::Device::Smartcard Class Reference

`#include <be_device_smartcard.h>`

Classes

- class **APDU**
- struct **APDUException**

  Exception thrown when a command fails.
- struct **APDUResponse**

  The data and status words returned by the card in response to a command.

Public Member Functions

- **Smartcard** (unsigned int cardNum)

  Connect to the Nth card in the system independent of any application installed on the card.
- **Smartcard** (unsigned int cardNum, const **Memory::uint8Array** &appID)

  Connect to the Nth card in the system and activate the application with the given identifier.
- **Memory::uint8Array getDedicatedFileObject** (const **Memory::uint8Array** &objectID)
- **APDUResponse sendAPDU** (**Device::Smartcard::APDU** &apdu)

  *Send an **APDU** (p. 281) to a card using the best transmission method available for the card.*
- **Memory::uint8Array getLastAPDU** () const
- **Memory::uint8Array getLastResponseData** () const
- std::string **getReaderID** () const

Obtain the identifier of the reader that the smartcard is plugged into.

- void **setDryrun** (bool state)
- ~**Smartcard** ()
- **Smartcard** (**Smartcard** &&other) noexcept

  Move constructor.
- **Smartcard** & **operator=** (**Smartcard** &&other) noexcept

  Move assignment.

H.141.1 Detailed Description

Representation of a single ISO 7816 smartcard in the system. A card can be associated with an application that is present on the card. Smartcards are accessed with a command/response protocol, and this class provides the capability to retrieve the response status and data whether the command succeeds or fails.

H.141.2 Constructor & Destructor Documentation

H.141.2.1 Smartcard() [1/3]

```
BiometricEvaluation::Device::Smartcard::Smartcard (
            unsigned int cardNum)
```

Connect to the Nth card in the system independent of any application installed on the card.

Cards are numbered according to reader sequencing. Therefore, the first card (number 0) is expected to be in the first reader.

Parameters

in	*cardNum*	The number of the card to attach to.

Exceptions

Error::ParameterError (p. 655)	No card exists for the given card number.
Error::StrategyError (p. 789)	Failed to access at least one of the readers.

H.141.2.2 Smartcard() [2/3]

```
BiometricEvaluation::Device::Smartcard::Smartcard (
            unsigned int cardNum,
            const  Memory::uint8Array & appID)
```

Connect to the Nth card in the system and activate the application with the given identifier.

Cards are numbered according to reader sequencing. Therefore, the first card (number 0) is expected to be in the first reader. The response data from application activation can be retrieved with the **getLastResponse↩ Data()** (p. 767) method.

Parameters

in	*cardNum*	The number of the card to attach to.
in	*appID*	The ID of the application to activate on the card.

Exceptions

APDUException (p. 283)	An error occurred activating the application. The status word fields on the exception's response
Error::ParameterError (p. 655)	No card exists for the given card number with the given appl
Error::StrategyError (p. 789)	Failed to access at least one of the readers.

H.141.2.3 ~Smartcard()

```
BiometricEvaluation::Device::Smartcard::~Smartcard ()
```

Destructor.

H.141.2.4 Smartcard() [3/3]

```
BiometricEvaluation::Device::Smartcard::Smartcard (
            Smartcard && other)  [noexcept]
```

Move constructor.

Smartcard (p. 764) objects are movable, maintaining the single instance of the access to the physical card. This allows the object to be placed in an STL container.

H.141.3 Member Function Documentation

H.141.3.1 getDedicatedFileObject()

```
Memory::uint8Array BiometricEvaluation::Device::Smartcard::getDedicatedFileObject (
          const  Memory::uint8Array & objectID)
```

Read a data object from the application dedicated file.

The objectID parameter must be a **TLV** (p. 813) octet string with the tag set to one of these values:

- 0x5C - A tag list data object.
- 0x5D - A header list data object.
- 0x4D - An extended header list data object.

Parameters

in	*objectID*	The ID of the requested object.

Returns

The dedicated file object.

Exceptions

APDUException *(p. 283)*	An error occurred activating the application. The status word fields on the exception's response
Error::StrategyError *(p. 789)*	An error occurred
Error::ParameterError *(p. 655)*	The

H.141.3.2 getLastAPDU()

`Memory::uint8Array BiometricEvaluation::Device::Smartcard::getLastAPDU () const`

Obtain a copy of the last **APDU** (p. 281) sent to the card.

Returns

The last sent **APDU** (p. 281) as an array of octets.

H.141.3.3 getLastResponseData()

`Memory::uint8Array BiometricEvaluation::Device::Smartcard::getLastResponseData () const`

Obtain a copy of the last response data returned from the card.

Returns

The last response data as an array of octets. May be empty.

H.141.3.4 getReaderID()

`std::string BiometricEvaluation::Device::Smartcard::getReaderID () const`

Obtain the identifier of the reader that the smartcard is plugged into.

Returns

The string identifier of the reader.

H.141.3.5 operator=()

```
Smartcard & BiometricEvaluation::Device::Smartcard::operator= (
            Smartcard && other)  [noexcept]
```

Move assignment.

Smartcard (p. 764) objects are movable, maintaining the single instance of the access to the physical card. This allows the object to be placed in an STL container.

H.141.3.6 sendAPDU()

```
APDUResponse BiometricEvaluation::Device::Smartcard::sendAPDU (
            Device::Smartcard::APDU & apdu)
```

Send an **APDU** (p. 281) to a card using the best transmission method available for the card.

Parameters

in,out	*apdu*	The **APDU** (p. 281) to be sent. Fields may be modified by the function, specifically the length field(s).

Exceptions

APDUException (p. 283)	The status words from the command response are something other than 0x9000. The status word
Error::StrategyError (p. 789)	

H.141.3.7 setDryrun()

```
void BiometricEvaluation::Device::Smartcard::setDryrun (
            bool state)
```

Set the 'dryrun' state.

Parameters

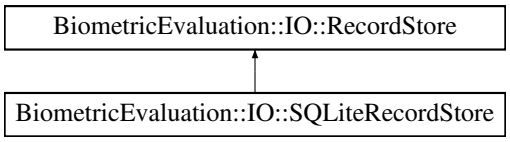
in	*state*	True when the **APDU** (p. 281) should be created, but not sent to the card. @seealso **get↩ Last↩ APDU()** (p. 767)

H.142 BiometricEvaluation::IO::SQLiteRecordStore Class Reference

An **IO::RecordStore** (p. 700) implementation using a SQLite database as the underlying record storage system.

`#include <be_io_sqliterecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::SQLiteRecordStore:



Public Member Functions

- **SQLiteRecordStore** (const std::string &pathname, const std::string &description)
- **SQLiteRecordStore** (const std::string &pathname, **IO::Mode** mode= **Mode::ReadOnly**)
- void **move** (const std::string &pathname) override

  *Move the **RecordStore*** (p. 700).
- void **sync** () const override
- unsigned int **getCount** () const override
- std::string **getPathname** () const override
- std::string **getDescription** () const override
- void **changeDescription** (const std::string &description) override
- uint64_t **getSpaceUsed** () const override

  Obtain real storage utilization.
- void **insert** (const std::string &key, const void *const data, const uint64_t size) override
- void **remove** (const std::string &key) override
- **Memory::uint8Array read** (const std::string &key) const override

Read a complete record from a store.

- uint64_t **length** (const std::string &key) const override
- void **flush** (const std::string &key) const override
- **RecordStore::Record sequence** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 700), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- **SQLiteRecordStore** (const **SQLiteRecordStore** &)=delete
- **SQLiteRecordStore** & **operator=** (const **SQLiteRecordStore** &)=delete
- virtual void **insert** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const **Memory::uint8Array** &data)
- virtual void **replace** (const std::string &key, const void *const data, const uint64_t size)

Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- virtual bool **containsKey** (const std::string &key) const

  *Determines whether the **RecordStore** (p. 700) contains an element with the specified key.*
- virtual **iterator begin** () noexcept
- virtual **iterator end** () noexcept

Additional Inherited Members

Public Types inherited from BiometricEvaluation::IO::RecordStore

- enum class **Kind** {
  BerkeleyDB , **Archive** , **File** , **SQLite** ,
  Compressed , **List** , **Default** = BerkeleyDB }
- using **Record** = struct Record
- using **iterator** = **IO::RecordStoreIterator**

Static Public Member Functions inherited from BiometricEvaluation::IO::RecordStore

- static bool **isRecordStore** (const std::string &pathname)

  *Determine if a location appears to be a **RecordStore** (p. 700).*
- static std::shared_ptr< **RecordStore** > **openRecordStore** (const std::string &pathname, **IO::Mode** mode= **Mode::ReadOnly**)

  *Open an existing **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static std::shared_ptr< **RecordStore** > **createRecordStore** (const std::string &pathname, const std::string &description, const **IO::RecordStore::Kind** &kind)

  *Create a new **RecordStore** (p. 700) and return a managed pointer to the the object representing that store.*
- static void **removeRecordStore** (const std::string &pathname)
- static void **mergeRecordStores** (const std::string &mergePathname, const std::string &description, const **IO::RecordStore::Kind** &kind, const std::vector< std::string > &pathnames, const std::function< bool()> &interrupt=[ ]() {return(false);})

  *Create a new **RecordStore** (p. 700) that contains the contents of several other RecordStores.*

Static Public Attributes inherited from BiometricEvaluation::IO::RecordStore

- static const std::string **INVALIDKEYCHARS**
- static const int **BE_RECSTORE_SEQ_START** = 1
- static const int **BE_RECSTORE_SEQ_NEXT** = 2

H.142.1 Detailed Description

An **IO::RecordStore** (p. 700) implementation using a SQLite database as the underlying record storage system.

H.142.2 Member Function Documentation

H.142.2.1 changeDescription()

```
void BiometricEvaluation::IO::SQLiteRecordStore::changeDescription (
            const std::string & description)  [override], [virtual]
```

Change the description of the **RecordStore** (p. 700).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 703).

H.142.2.2 flush()

```
void BiometricEvaluation::IO::SQLiteRecordStore::flush (
            const std::string & key) const  [override], [virtual]
```

Commit the record's data to storage.

Parameters

in	*key*	The key of the record to be flushed.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 704).

H.142.2.3 getCount()

`unsigned int BiometricEvaluation::IO::SQLiteRecordStore::getCount () const [override], [virtual]`

Obtain the number of items in the **RecordStore** (p. 700).

Returns

The number of items in the **RecordStore** (p. 700).

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.142.2.4 getDescription()

`std::string BiometricEvaluation::IO::SQLiteRecordStore::getDescription () const [override], [virtual]`

Obtain a textual description of the **RecordStore** (p. 700).

Returns

The **RecordStore** (p. 700)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.142.2.5 getPathname()

`std::string BiometricEvaluation::IO::SQLiteRecordStore::getPathname () const [override], [virtual]`

Return the path name of the **RecordStore** (p. 700).

Returns

Where in the file system the **RecordStore** (p. 700) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 705).

H.142.2.6 getSpaceUsed()

`uint64_t BiometricEvaluation::IO::SQLiteRecordStore::getSpaceUsed () const [override], [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** (p. 700).

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 706).

H.142.2.7 insert() [1/2]

```
virtual void BiometricEvaluation::IO::RecordStore::insert (
            const std::string & key,
            const  Memory::uint8Array & data)  [virtual]
```

Insert a record into the store.

Parameters

in	*key*	The key of the record to be inserted.
in	*data*	The data for the record.

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 706).

H.142.2.8 insert() [2/2]

```
void BiometricEvaluation::IO::SQLiteRecordStore::insert (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [override], [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.

Parameters

in	*size*	The size of the record, in bytes.

Exceptions

Error::ObjectExists (p. 637)	A record with the given key is already present.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underlying st

Implements **BiometricEvaluation::IO::RecordStore** (p. 707).

H.142.2.9 length()

```
uint64_t BiometricEvaluation::IO::SQLiteRecordStore::length (
            const std::string & key) const  [override], [virtual]
```

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 708).

H.142.2.10 move()

```
void BiometricEvaluation::IO::SQLiteRecordStore::move (
            const std::string & pathname)  [override], [virtual]
```

Move the **RecordStore** (p. 700).

The **RecordStore** (p. 700) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 700).

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 710).

H.142.2.11 read()

```
Memory::uint8Array BiometricEvaluation::IO::SQLiteRecordStore::read (
            const std::string & key) const  [override], [virtual]
```

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

in	*key*	The key of the record to be read.

Returns

The record associated with the key.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	A record for the key does not exist.
Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 712).

H.142.2.12 remove()

```
void BiometricEvaluation::IO::SQLiteRecordStore::remove (
            const std::string & key)  [override], [virtual]
```

Remove a record from the store.

Parameters

in	*key*	The key of the record to be removed.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 713).

H.142.2.13 replace() [1/2]

```
virtual void BiometricEvaluation::IO::RecordStore::replace (
            const std::string & key,
            const  Memory::uint8Array & data)  [virtual]
```

Replace a complete record in a **RecordStore** (p. 700).

Parameters

in	*key*	The key of the record to be replaced.
in	*data*	The data for the record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.142.2.14 replace() [2/2]

```
virtual void BiometricEvaluation::IO::RecordStore::replace (
            const std::string & key,
            const void *const data,
            const uint64_t size)  [virtual]
```

Replace a complete record in a **RecordStore** (p. 700).

Parameters

in	*key*	The key of the record to be replaced.
in	*data*	The data for the record.
in	*size*	The size of the record, in bytes.

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	The **RecordStore** (p. 700) is opened read-only, or an error occurred when using the underl

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 714).

H.142.2.15 sequence()

```
RecordStore::Record BiometricEvaluation::IO::SQLiteRecordStore::sequence (
            int cursor = BE_RECSTORE_SEQ_NEXT)  [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key/data pairs.

Sequencing means to start at some point in the store and return the record, then repeatedly calling the function to return the next record. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

in	*cursor*	The location within the sequence of the key/-data pair to return.

Returns

The record that is currently in sequence.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 715).

H.142.2.16 sequenceKey()

```
std::string BiometricEvaluation::IO::SQLiteRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT) [override], [virtual]
```

Sequence through a **RecordStore** (p. 700), returning the key.

Sequencing means to start at some point in the store and return the key, then repeatedly calling the function to return the next key. The starting point is typically the first record, and is set to that when the **RecordStore** (p. 700) object is created. The starting point can be reset by calling this method with the cursor parameter set to BE_RECSTORE_SEQ_START.

Parameters

in	*cursor*	The location within the sequence of the key/-data pair to return.

Returns

The key of the currently sequenced record.

Exceptions

Error::ObjectDoesNotExist (p. 637)	End of sequencing.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 716).

H.142.2.17 setCursorAtKey()

```
void BiometricEvaluation::IO::SQLiteRecordStore::setCursorAtKey (
            const std::string & key)  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 700), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 777).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 777).

Exceptions

Error::ObjectDoesNotExist (p. 637)	A record for the key does not exist.
Error::StrategyError (p. 789)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.142.2.18 sync()

```
void BiometricEvaluation::IO::SQLiteRecordStore::sync () const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 717).

H.143 BiometricEvaluation::Process::Statistics Class Reference

The **Statistics** (p. 780) class provides an interface for gathering process statistics, such as memory usage, system time, etc.

`#include <be_process_statistics.h>`

Public Member Functions

- **Statistics** ()
- **Statistics** (const std::shared_ptr< **IO::FileLogCabinet** > &logCabinet, bool doTasksLogging=false)
- **Statistics** (const std::shared_ptr< **IO::Logsheet** > &logSheet, std::optional< std::shared_ptr< **IO::Logsheet** > > tasksLogSheet=std::nullopt)

  Construct a Statistic object that logs to an existing Logsheet.
- std::tuple< uint64_t, uint64_t > **getCPUTimes** ()
- std::vector< std::tuple< pid_t, float, float > > **getTasksStats** ()
- std::tuple< uint64_t, uint64_t, uint64_t, uint64_t, uint64_t > **getMemorySizes** ()
- uint32_t **getNumThreads** ()
- void **logStats** ()

  Create a snapshot of the current process statistics in the FileLogsheet created in the FileLogCabinet.
- std::string **getComment** () const

  Get the comment that is appended to every auto logger entry.
- void **setComment** (std::string_view comment)

  Set a comment for each log entry.
- void **startAutoLogging** (std::chrono::microseconds interval)
- void **startAutoLogging** (uint64_t interval)
- void **stopAutoLogging** ()

H.143.1 Detailed Description

The **Statistics** (p. 780) class provides an interface for gathering process statistics, such as memory usage, system time, etc.

There are two groups of statistics: **Memory** (p. 156) and time info for the process, and system/user time for all tasks (threads) created by the process. The information gathered by objects of this class are for the current process, and can automatically be logged to a FileLogsheet object contained within the provided FileLog↩ Cabinet. The task statistics are optionally logged.

Note

The resolution of a returned value for many methods may not match the resolution allowed by the interface. For example, the operating system my allow for second resolution whereas the interface allows microsecond resolution.

H.143.2 Constructor & Destructor Documentation

H.143.2.1 Statistics() [1/3]

```
BiometricEvaluation::Process::Statistics::Statistics ()
```

Construct a **Statistics** (p. 780) object without logging, for clients to obtain process statistics directly.

H.143.2.2 Statistics() [2/3]

```
BiometricEvaluation::Process::Statistics::Statistics (
            const std::shared_ptr< IO::FileLogCabinet > & logCabinet,
            bool doTasksLogging = false)
```

Construct a **Statistics** (p. 780) object with the associated FileLogCabinet.

Parameters

in	*logCabinet*	The FileLogCabinet obejct where this object will create a FileLogsheet to contain the statistic information for the process.

Parameters

in	*doTasksLogging*	If true, create a second log sheet containing information for each task owned by the PID.

Exceptions

Error::NotImplemented (p. 636)	Logging is not supported on this OS. This exception can be thrown when any portion of the sta
Error::ObjectExists (p. 637)	The FileLogsheet already exists. This exception should rarely, if e
Error::StrategyError (p. 789)	Failure to create the FileLogsheet in the cabinet.

H.143.2.3 Statistics() [3/3]

```
BiometricEvaluation::Process::Statistics::Statistics (
            const std::shared_ptr< IO::Logsheet > & logSheet,
            std::optional< std::shared_ptr< IO::Logsheet > > tasksLogSheet = std::nullopt)
```

Construct a Statistic object that logs to an existing Logsheet.

Parameters

in	*logSheet*	Existing Logsheet that will be appended.

Parameters

	tasksLogSheet	Optional log sheet that will contain information for each task owned by the PID.

Exceptions

Error::NotImplemented (p. 636)	Logging is not supported on this OS. This exception can be thrown when any portion of the sta

H.143.3 Member Function Documentation

H.143.3.1 getComment()

`std::string BiometricEvaluation::Process::Statistics::getComment () const`

Get the comment that is appended to every auto logger entry.

Returns

The comment string.

H.143.3.2 getCPUTimes()

`std::tuple< uint64_t, uint64_t > BiometricEvaluation::Process::Statistics::getCPUTimes ()`

Obtain the total user and system times for the process, in microseconds.

An example call:

```
uint64_t utime, stime;
std::tie(utime, stime) = stats.getCPUTimes();
```

Note

This method may not be implemented in all operating systems.

Returns

A std::tuple<> containing user time, system time.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when obtaining the process statistics from the operating system. The except
Error::NotImplemented *(p. 636)*	This method is not implemented on this OS.

H.143.3.3 getMemorySizes()

```
std::tuple< uint64_t, uint64_t, uint64_t, uint64_t, uint64_t > BiometricEvaluation::Process::↩
Statistics::getMemorySizes ()
```

Obtain the current virtual memory (VM) set sizes for the process, in kilobytes. An example call:

```
uint64_t vmrss, vmsize, vmpeak, vmdata, vmstack;
std::tie(vmrss, vmsize, vmpeak, vmdata, vmstack)
  = stats.getMemorySizes();
```

Note

This method may not be implemented in all operating systems.

Returns

A std::tuple<> containing VM resident size, VM size, VM peak, VM data size, VM stack size.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when obtaining the process statistics from the operating system. The except
Error::NotImplemented *(p. 636)*	This method is not implemented on this OS.

H.143.3.4 getNumThreads()

```
uint32_t BiometricEvaluation::Process::Statistics::getNumThreads ()
```

Obtain the number of threads composing this process.

Note

This method may not be implemented in all operating systems.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when obtaining the process info from the operating system. The exception i
Error::NotImplemented *(p. 636)*	This method is not implemented on this OS.

H.143.3.5 getTasksStats()

```
std::vector< std::tuple< pid_t, float, float > > BiometricEvaluation::Process::Statistics↩
::getTasksStats ()
```

Obtain the current child tasks statistics for the process. The time values are in units of seconds.

An example call and processing: auto allStats = stats.getTasksStats(); for (auto [tid, utime, stime]: allStats) { cout << "TID is " << tid << " utime is " << utime << ", stime is " << stime << '
'; }

Note

This method may not be implemented in all operating systems.

Returns

A std::vector<> containing std::tuple<> elements containing Task ID, user time, system time.

Exceptions

Error::StrategyError (p. 789)	An error occurred when obtaining the process statistics from the operating system. The except
Error::NotImplemented (p. 636)	This method is not implemented on this OS.

H.143.3.6 logStats()

```
void BiometricEvaluation::Process::Statistics::logStats ()
```

Create a snapshot of the current process statistics in the FileLogsheet created in the FileLogCabinet.

Exceptions

Error::ObjectDoesNotExist (p. 637)	The FileLogsheet does not exist; this object was not created with FileLogCabinet object.
Error::StrategyError (p. 789)	An error occurred when writing to the FileLogsheet.
Error::NotImplemented (p. 636)	The statistics gathering is not implemented for this operating system.

H.143.3.7 setComment()

```
void BiometricEvaluation::Process::Statistics::setComment (
            std::string_view comment)
```

Set a comment for each log entry.

The comment string is auto-appended to the end of each log entry.

Parameters

comment	The comment string.

H.143.3.8 startAutoLogging()

```
void BiometricEvaluation::Process::Statistics::startAutoLogging (
            std::chrono::microseconds interval)
```

Start auto logging process statistics.

Parameters

interval	The time gap between the capture of the statistics in microseconds.

H.143.3.9 stopAutoLogging()

```
void BiometricEvaluation::Process::Statistics::stopAutoLogging ()
```

Stop auto logging process statistics.

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	Not currently logging.

H.144 BiometricEvaluation::Framework::Status Class Reference

```
#include <be_framework_status.h>
```

Public Types

- enum class **Type** { **Debug** , **Warning** , **Error** }

Public Member Functions

- **Status** (**Type** type, const std::string &message, const std::string &identifier="")
  Status *(p. 786) constructor.*
- **Type** **getType** () const noexcept
  Obtain the Type of this ***Status*** *(p. 786)' message.*
- std::string **getMessage** () const noexcept
  Obtain the explanatory message from this ***Status*** *(p. 786).*
- std::string **getIdentifier** () const noexcept
  Obtain the identifier from this ***Status*** *(p. 786).*

H.144.1 Detailed Description

Information communicated back from framework methods.

H.144.2 Member Enumeration Documentation

H.144.2.1 Type

```
enum class BiometricEvaluation::Framework::Status::Type [strong]
```

Type of status received.

Enumerator

Debug	Informational/debugging. Processing should continue.
Warning	Something seems off about the operation, but the output might be fine.
Error	Processing absolutely should stop.

H.144.3 Constructor & Destructor Documentation

H.144.3.1 Status()

```
BiometricEvaluation::Framework::Status::Status (
                Type type,
              const std::string & message,
              const std::string & identifier = "")
```

Status (p. 786) constructor.

Parameters

code	Return code from a function or method.
message	Message providing insight into code's value.

H.144.4 Member Function Documentation

H.144.4.1 getIdentifier()

`std::string BiometricEvaluation::Framework::Status::getIdentifier () const [inline], [noexcept]`

Obtain the identifier from this **Status** (p. 786).

The identifier is used to provide more context about the message and is user-defined.

Returns

Identifier associated with this **Status** (p. 786).

Note

May be empty.

H.144.4.2 getMessage()

`std::string BiometricEvaluation::Framework::Status::getMessage () const [inline], [noexcept]`

Obtain the explanatory message from this **Status** (p. 786).

Returns

Explanatory message.

Note

May be empty.

H.144.4.3 getType()

`Type BiometricEvaluation::Framework::Status::getType () const [inline], [noexcept]`

Obtain the Type of this **Status** (p. 786)' message.

Returns

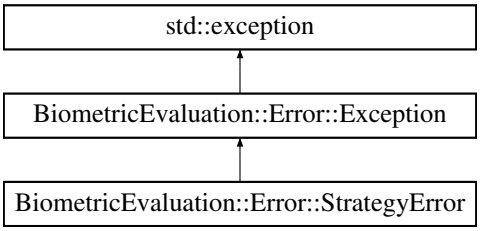
Type of status

H.145 BiometricEvaluation::Error::StrategyError Class Reference

A **StrategyError** (p. 789) 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** (const std::string &info)

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char * **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.145.1 Detailed Description

A **StrategyError** (p. 789) object is thrown when the underlying implementation of this interface encounters an error.

H.145.2 Constructor & Destructor Documentation

H.145.2.1 StrategyError() [1/2]

```
BiometricEvaluation::Error::StrategyError::StrategyError ()
```

Construct a **StrategyError** (p. 789) object with the default information string.

H.145.2.2 StrategyError() [2/2]

```
BiometricEvaluation::Error::StrategyError::StrategyError (
            const std::string & info)
```

Construct a **StrategyError** (p. 789) object with an information string appended to the default information string.

H.146 BiometricEvaluation::Video::Stream Class Reference

Public Member Functions

- virtual float **getFPS** ()=0

Obtain the average frame rate of the video stream.

- virtual uint64_t **getFrameCount** ()=0

  Obtain the number of frames in the video stream.

- virtual **Video::Frame** **getFrame** (uint32_t frameNum)=0

  Obtain a frame from the video stream.

- virtual std::vector< **Video::Frame** > **getFrameSequence** (int64_t startTime, int64_t endTime)=0

  Obtain a sequence of frames from the video stream.

- virtual void **setFrameScale** (float xScale, float yScale)=0

  Set the scaling factors for returned video frames.

- virtual void **setFramePixelFormat** (const **Image::PixelFormat** pixelFormat)=0

  Set the pixel format for returned video frames.

H.146.1 Member Function Documentation

H.146.1.1 getFPS()

```
virtual float BiometricEvaluation::Video::Stream::getFPS ()  [pure virtual]
```

Obtain the average frame rate of the video stream.

Returns

The average frame rate. A value of 0 means the frame rate cannot be determined.

H.146.1.2 getFrame()

```
virtual  Video::Frame BiometricEvaluation::Video::Stream::getFrame (
            uint32_t frameNum)  [pure virtual]
```

Obtain a frame from the video stream.

Parameters

frameNum	**Frame** (p. 464) number, >= 1

Exceptions

Error::ParameterError *(p. 655)*	frameNum is too large.
Error::StrategyError *(p. 789)*	No codec available for the video stream or other failure to read the stream.

H.146.1.3 getFrameCount()

```
virtual uint64_t BiometricEvaluation::Video::Stream::getFrameCount ()  [pure virtual]
```

Obtain the number of frames in the video stream.

Returns

The number of frames in the stream; will be 0 if unknown.

H.146.1.4 getFrameSequence()

```
virtual std::vector< Video::Frame > BiometricEvaluation::Video::Stream::getFrameSequence (
            int64_t startTime,
            int64_t endTime) [pure virtual]
```

Obtain a sequence of frames from the video stream.

The end time can be greater than the length of the stream, and is not considered an error. Frames up to and including the last will be returned.

Parameters

startTime	Approximate time of the starting frame, millisecconds.
endTime	Approximate time of the ending frame, millisecconds

Exceptions

Error::StrategyError *(p. 789)*	No codec available for the video stream or other failure to read the stream.

H.146.1.5 setFramePixelFormat()

```
virtual void BiometricEvaluation::Video::Stream::setFramePixelFormat (
            const Image::PixelFormat pixelFormat) [pure virtual]
```

Set the pixel format for returned video frames.

Parameters

pixelFormat	The pixel format of all returned frames.

H.146.1.6 setFrameScale()

```
virtual void BiometricEvaluation::Video::Stream::setFrameScale (
            float xScale,
            float yScale)  [pure virtual]
```

Set the scaling factors for returned video frames.

Parameters

xScale	The scaling factor for frame width.
yScale	The scaling factor for frame height.

H.147 BiometricEvaluation::Feature::AN2K11EFS::Substrate Struct Reference

```
#include <be_feature_an2k11efs.h>
```

Public Attributes

- bool **present** {false}
- SubstrateCode **cls** {SubstrateCode::Unknown}
- std::string **osd** {}

H.147.1 Detailed Description

Description of surface on which latent was deposited

H.147.2 Member Data Documentation

H.147.2.1 cls

```
SubstrateCode BiometricEvaluation::Feature::AN2K11EFS::Substrate::cls {SubstrateCode::Unknown}
```

Type of substrate (required)

H.147.2.2 osd

```
std::string BiometricEvaluation::Feature::AN2K11EFS::Substrate::osd {}
```

Description and/or clarification (optional)

H.147.2.3 present

`bool BiometricEvaluation::Feature::AN2K11EFS::Substrate::present {false}`

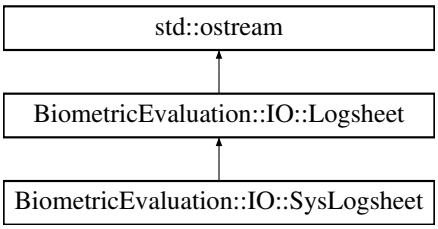
Whether this field was present

H.148 BiometricEvaluation::IO::SysLogsheet Class Reference

A class to represent a single logging mechanism to a logging service on the network.

`#include <be_io_syslogsheet.h>`

Inheritance diagram for BiometricEvaluation::IO::SysLogsheet:



Public Member Functions

- **SysLogsheet** (const std::string &url, const std::string &description, const std::string &appname, bool sequenced, bool utc)

  Create a new log sheet.
- **SysLogsheet** (const std::string &url, const std::string &description, const std::string &appname, const std::string &hostname, bool sequenced, bool utc)

  Create a new log sheet.
- ~**SysLogsheet** ()
- void **write** (const std::string &entry)

  Write a string as an entry to the backing store.
- void **writeComment** (const std::string &entry)

  Write a string as a comment to the backing store.
- void **writeDebug** (const std::string &entry)

  Write a string as a debug entry to the backing store.
- void **sync** ()

  Synchronize any buffered data to the underlying backing store.

Public Member Functions inherited from BiometricEvaluation::IO::Logsheet

- **Logsheet** ()

  *Create a **Logsheet** (p. 585) that has no backing store. A log entry is maintained, but cannot be permanently stored. This is the Null **Logsheet** (p. 585).*
- virtual ~**Logsheet** ()
- void **newEntry** ()

  Start a new entry, causing the existing entry to be closed and written.
- std::string **getCurrentEntry** () const

  Obtain the contents of the current entry currently under construction.
- void **resetCurrentEntry** ()

- uint32_t **getCurrentEntryNumber** () const

  Obtain the current entry number.
- void **setCommit** (const bool state)

  Enable or disable the commitment of normal entries to the backing log storage.
- bool **getCommit** () const

  Get the current entry commit state.
- void **setDebugCommit** (const bool state)

  Enable or disable the commitment of debug entries to the backing log storage.
- bool **getDebugCommit** () const

  Get the current debug entry commit state.
- void **setCommentCommit** (const bool state)

  Enable or disable the commitment of comment entries to the backing log storage.
- bool **getCommentCommit** () const

  Get the current comment entry commit state.
- void **setAutoSync** (bool state)
- bool **getAutoSync** () const

Protected Member Functions

- **SysLogsheet** (const **SysLogsheet** &)
- **SysLogsheet** & **operator=** (const **SysLogsheet** &)
- void **setup** (const std::string &url, const std::string &description)
- void **writeToLogger** (const std::string &priority, const char delimiter, const std::string &prefix, const std::string &message)

Protected Member Functions inherited from BiometricEvaluation::IO::Logsheet

- void **incrementEntryNumber** ()

  Increment the current entry number.
- std::string **getCurrentEntryNumberAsString** () const

  Obtain the current entry 'tag', in 'Edddd' format.

Protected Attributes

- std::string **_hostname**
- std::string **_appname**
- std::string **_procid**
- int **_sockFD**
- bool **_sequenced**
- bool **_operational**
- bool **_utc**

Additional Inherited Members

Public Types inherited from BiometricEvaluation::IO::Logsheet

- enum class **Kind** { **Null** , **File** , **Syslog** }

Static Public Member Functions inherited from BiometricEvaluation::IO::Logsheet

- static **Logsheet::Kind** **getTypeFromURL** (const std::string &url)

  *Map the URL scheme, taken from a string containing the entire URL, into a **Logsheet** (p. 585) type.*
- static bool **lineIsEntry** (const std::string &line)

  Helper function to determine whether a string is a valid log entry.
- static bool **lineIsComment** (const std::string &line)

  Helper function to determine whether a string is a valid comment log entry.
- static bool **lineIsDebug** (const std::string &line)

  Helper function to determine whether a string is a valid debug log entry.
- static std::string **trim** (const std::string &entry)

  *Trim delimiters from **Logsheet** (p. 585) entries.*

Static Public Attributes inherited from BiometricEvaluation::IO::Logsheet

- static const char **CommentDelimiter** = '#'
- static const char **EntryDelimiter** = 'E'
- static const char **DebugDelimiter** = 'D'
- static const std::string **DescriptionTag**
- static const std::string **FILEURLSCHEME**
- static const std::string **SYSLOGURLSCHEME**

H.148.1 Detailed Description

A class to represent a single logging mechanism to a logging service on the network.

Log entries are sent to the logging server in RFC5424 format with a timestamp of the local system in UTC. Normal and comment entries are sent to the logger with a PRI field indicating the 'local0' facility and a severity of 'Informational'. Debug entries are sent with facility of 'local1' and severity 'Debug'. A basic syslog config file would contain these lines: local0.info /var/log/info.log local1.debug /var/log/debug.log

The hostname is added to each entry but may be overridden by constructing the object with a given hostname, including the RFC5424 NILVALUE character. The PROCID part of each log message will be filled in with the process ID. Multi-line messages are segmented and sent the to logger as separate entries with the same timestamp and sequence number.

H.148.2 Constructor & Destructor Documentation

H.148.2.1 SysLogsheet() [1/3]

```
BiometricEvaluation::IO::SysLogsheet::SysLogsheet (
            const std::string & url,
            const std::string & description,
            const std::string & appname,
            bool sequenced,
            bool utc)
```

Create a new log sheet.

Parameters

in	*url*	The Uniform Resource Locator describing the logging service. Accepted forms are syslog↩ ://hostname↩ :port
in	*description*	The text used to describe the sheet. This text is written into the log prior to any entries.
in	*appname*	The name of the application. This text is written into each log entry.

Parameters

in	*sequenced*	True if each entry should include a sequence number, false if not.
in	*utc*	True if timestamps should be in Coordinated Universal **Time** (p. 185) (UTC), false for local time.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when connecting to the logging system, or URL is malformed.

H.148.2.2 SysLogsheet() [2/3]

```
BiometricEvaluation::IO::SysLogsheet::SysLogsheet (
            const std::string & url,
            const std::string & description,
            const std::string & appname,
            const std::string & hostname,
            bool sequenced,
            bool utc)
```

Create a new log sheet.

Parameters

in	*url*	The Uniform Resource Locator describing the logging service. Accepted forms are syslog↩ ://hostname↩ :port
in	*description*	The text used to describe the sheet. This text is written into the log prior to any entries.
in	*appname*	The name of the application. This text is written into each log entry.

Parameters

in	*hostname*	The string to use as the hostname for all log entries.
in	*sequenced*	True if each entry should include a sequence number, false if not.
in	*utc*	True if timestamps should be in Coordinated Universal **Time** (p. 185) (UTC), false for local time.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when connecting to the logging system, or URL is malformed.

H.148.2.3 ~SysLogsheet()

`BiometricEvaluation::IO::SysLogsheet::~SysLogsheet ()`

Destructor

H.148.2.4 SysLogsheet() [3/3]

```
BiometricEvaluation::IO::SysLogsheet::SysLogsheet (
            const SysLogsheet & ) [protected]
```

Prevent copying of **SysLogsheet** (p. 793) objects

H.148.3 Member Function Documentation

H.148.3.1 operator=()

```
SysLogsheet & BiometricEvaluation::IO::SysLogsheet::operator= (
            const SysLogsheet & ) [protected]
```

Prevent copying of **SysLogsheet** (p. 793) objects

H.148.3.2 setup()

```
void BiometricEvaluation::IO::SysLogsheet::setup (
            const std::string & url,
            const std::string & description) [protected]
```

Helper function to build connections

H.148.3.3 sync()

```
void BiometricEvaluation::IO::SysLogsheet::sync () [virtual]
```

Synchronize any buffered data to the underlying backing store.

This syncing is dependent on the behavior of the underlying storage mechanism.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 593).

H.148.3.4 write()

```
void BiometricEvaluation::IO::SysLogsheet::write (
            const std::string & entry) [virtual]
```

Write a string as an entry to the backing store.

This does not affect the current log entry buffer, but does increment the entry number.

Parameters

in	*entry*	The text of the log entry.

Exceptions

Error::StrategyError (p. 789)	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 594).

H.148.3.5 writeComment()

```
void BiometricEvaluation::IO::SysLogsheet::writeComment (
            const std::string & entry)  [virtual]
```

Write a string as a comment to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

in	*entry*	The text of the comment.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 594).

H.148.3.6 writeDebug()

```
void BiometricEvaluation::IO::SysLogsheet::writeDebug (
            const std::string & entry)  [virtual]
```

Write a string as a debug entry to the backing store.

This does not affect the current log entry buffer, and does not increment the entry number. A debug line is prefixed with DebugDelimiter followed by a space.

Parameters

in	*entry*	The text of the debug message.

Exceptions

Error::StrategyError *(p. 789)*	An error occurred when logging.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 595).

H.148.3.7 writeToLogger()

```
void BiometricEvaluation::IO::SysLogsheet::writeToLogger (
            const std::string & priority,
            const char delimiter,
            const std::string & prefix,
            const std::string & message)  [protected]
```

Helper function to write to the logger

H.148.4 Member Data Documentation

H.148.4.1 _operational

```
bool BiometricEvaluation::IO::SysLogsheet::_operational  [protected]
```

Whether the sheet is operational

H.148.4.2 _sequenced

```
bool BiometricEvaluation::IO::SysLogsheet::_sequenced  [protected]
```

Whether to include entry sequence numbers

H.148.4.3 _sockFD

```
int BiometricEvaluation::IO::SysLogsheet::_sockFD  [protected]
```

Socket file descriptor for the logging system

H.148.4.4 _utc

```
bool BiometricEvaluation::IO::SysLogsheet::_utc  [protected]
```

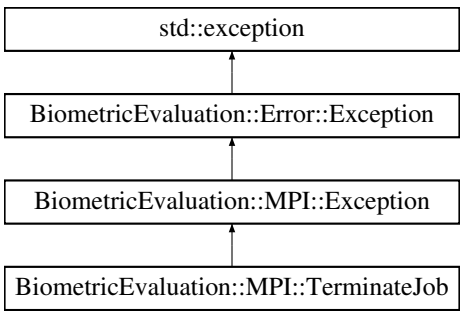
Whether time stamps are in UTC

H.149 BiometricEvaluation::MPI::TerminateJob Class Reference

An exception that when thrown from a Task should result in the entire job (all tasks) being shut down by the **Distributor** (p. 405).

```
#include <be_mpi_exception.h>
```

Inheritance diagram for BiometricEvaluation::MPI::TerminateJob:



Public Member Functions

- **TerminateJob** ()
- **TerminateJob** (std::string info)

  Constructor.

Public Member Functions inherited from BiometricEvaluation::MPI::Exception

- **Exception** ()
- **Exception** (std::string info)

  Constructor.
- virtual ∼**Exception** () noexcept=default

Public Member Functions inherited from BiometricEvaluation::Error::Exception

- **Exception** ()
- **Exception** (std::string info)
- const char ∗ **what** () const noexcept override
- const std::string **whatString** () const noexcept

H.149.1 Detailed Description

An exception that when thrown from a Task should result in the entire job (all tasks) being shut down by the **Distributor** (p. 405).

H.149.2 Constructor & Destructor Documentation

H.149.2.1 TerminateJob() [1/2]

```
BiometricEvaluation::MPI::TerminateJob::TerminateJob ()
```

Construct with default information string.

H.149.2.2 TerminateJob() [2/2]

```
BiometricEvaluation::MPI::TerminateJob::TerminateJob (
            std::string info)
```

Constructor.

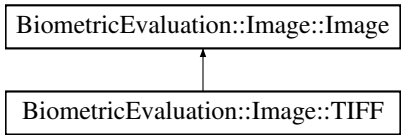
Parameters

info	Custom information string. Will be appended to the default information string.

H.150 BiometricEvaluation::Image::TIFF Class Reference

`#include <be_image_tiff.h>`

Inheritance diagram for BiometricEvaluation::Image::TIFF:



Classes

- struct **ClientIO**

Public Member Functions

- **TIFF** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **TIFF** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Memory::uint8Array getRawData** () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::uint8Array getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.

Public Member Functions inherited from BiometricEvaluation::Image::Image

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm getCompressionAlgorithm** () const

  Accessor for the CompressionAlgorithm of the image.
- **Resolution getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array getData** () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual **Memory::uint8Array getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Size getDimensions** () const

Accessor for the dimensions of the image in pixels.

- uint32_t **getColorDepth** () const

  Accessor for the color depth of the image in bits.

- uint16_t **getBitDepth** () const

  Accessor for the number of bits per color component.

- bool **hasAlphaChannel** () const

  Accessor for the presence of an alpha channel.

- statusCallback_t **getStatusCallback** () const

  Get handle to status callback function.

- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Static Public Member Functions

- static bool **isTIFF** (const uint8_t *data, const uint64_t size)

  *Determine if image is encoded as **TIFF*** (p. 804).

- static bool **isTIFF** (const **Memory::uint8Array** &data)

  *Determine if image is encoded as **TIFF*** (p. 804).

- static std::string **libtiffMessageToString** (const char *module, const char *format, va_list args)

  Convert libtiff message to string.

Static Public Member Functions inherited from BiometricEvaluation::Image::Image

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.

- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image*** (p. 477) *object.*

- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image*** (p. 477) *object.*

- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of an image file and create an **Image*** (p. 477) *object.*

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.

- static **BiometricEvaluation::Image::Raw** **getRawImage** (const std::shared_ptr< **BiometricEvaluation::Image::Image** > &image)

  *Obtain **Image::Raw*** (p. 688) *version of an **Image::Image*** (p. 477).

- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::Image::Image

- using **statusCallback_t**

Protected Member Functions inherited from BiometricEvaluation::Image::Image

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .
- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.
- void **setColorDepth** (const uint32_t colorDepth)

  Mutator for the color depth of the image in bits.
- void **setBitDepth** (const uint16_t bitDepth)

  Mutator for the number of bits per component for color components in the image, in bits.
- const uint8_t * **getDataPointer** () const
- uint64_t **getDataSize** () const
- void **setHasAlphaChannel** (const bool **hasAlphaChannel**)

  Mutator for the presence of an alpha channel.

H.150.1 Detailed Description

A TIFF-encoded image.

H.150.2 Member Function Documentation

H.150.2.1 getRawData()

```
Memory::uint8Array BiometricEvaluation::Image::TIFF::getRawData () const  [virtual]
```

Accessor for the raw image data. The data returned should not be compressed or encoded.

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 486).

H.150.2.2 getRawGrayscaleData()

```
Memory::uint8Array BiometricEvaluation::Image::TIFF::getRawGrayscaleData (
            uint8_t depth) const  [virtual]
```

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 16, 8, or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError (p. 390)	**Error** (p. 112) decompressing image data.
Error::NotImplemented (p. 636)	Unsupported conversion based on source color depth.
Error::ParameterError (p. 655)	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implements **BiometricEvaluation::Image::Image** (p. 487).

H.150.2.3 isTIFF() [1/2]

```
static bool BiometricEvaluation::Image::TIFF::isTIFF (
            const Memory::uint8Array & data) [static]
```

Determine if image is encoded as **TIFF** (p. 804).

Parameters

in	*data*	**Image** (p. 477) data.

Returns

true if data appears to be encoded with **TIFF** (p. 804), false otherwise.

H.150.2.4 isTIFF() [2/2]

```
static bool BiometricEvaluation::Image::TIFF::isTIFF (
            const uint8_t * data,
            const uint64_t size)  [static]
```

Determine if image is encoded as **TIFF** (p. 804).

Parameters

in	*data*	**Image** (p. 477) data.
in	*size*	**Size** (p. 763) of data.

Returns

true if data appears to be encoded with **TIFF** (p. 804), false otherwise.

H.150.2.5 libtiffMessageToString()

```
static std::string BiometricEvaluation::Image::TIFF::libtiffMessageToString (
            const char * module,
            const char * format,
            va_list args)  [static]
```

Convert libtiff message to string.

Parameters

in	*module*	libtiff module with an error.
in	*format*	printf(3)-style format string.

Parameters

in	*args*	printf(3)-style arguments.

Returns

Message containing parameters.

H.151 BiometricEvaluation::Time::Timer Class Reference

This class can be used by applications to report the amount of time a block of code takes to execute.

`#include <be_time_timer.h>`

Public Types

- using **BE_CLOCK_TYPE**

Public Member Functions

- **Timer** ()
- **Timer** (const std::function< void()> &func)

  Construct a timer and time a function immediately.
- void **start** ()

  Start tracking time.
- void **stop** ()

  Stop tracking time.
- template<typename Duration >
  std::uintmax_t **elapsed** () const

  *Get the elapsed time between calls to this object's **start()** (p. 812) and **stop()** (p. 812) methods.*
- template<typename Duration >
  std::string **elapsedStr** (bool displayUnits=false) const

  Convenience method for printing elapsed time as a string.
- std::common_type_t< BE_CLOCK_TYPE::time_point::duration, BE_CLOCK_TYPE::time_point::duration > **elapsedTimePoint** () const

  *Get the elapsed time between calls to this object's **start()** (p. 812) and **stop()** (p. 812) methods.*
- **Timer** & **time** (const std::function< void()> &func)

  Record the runtime of a function.

Static Public Member Functions

- template<typename Duration >
  static std::string **units** ()

H.151.1 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()** (p. 812) and **Timer::stop()** (p. 812) calls, then use **Timer::elapsed()** (p. 811) to obtain the calculated time of the operation.

Warning

Timers are not threadsafe and should only be used to time operations within the same thread.

H.151.2 Member Typedef Documentation

H.151.2.1 BE_CLOCK_TYPE

```
using BiometricEvaluation::Time::Timer::BE_CLOCK_TYPE
```

Initial value:

```
std::chrono::steady_clock
```

Clock type to use, aliased for easy replacement.

H.151.3 Constructor & Destructor Documentation

H.151.3.1 Timer() [1/2]

```
BiometricEvaluation::Time::Timer::Timer ()
```

Constructor for the **Timer** (p. 809) object.

H.151.3.2 Timer() [2/2]

```
BiometricEvaluation::Time::Timer::Timer (
            const std::function< void()> & func)
```

Construct a timer and time a function immediately.

Parameters

func	A function to time immediately.

Exceptions

Error::StrategyError (p. 789)	Propagated from **time()** (p. 812).

H.151.4 Member Function Documentation

H.151.4.1 elapsed()

```
template<typename Duration >
std::uintmax_t BiometricEvaluation::Time::Timer::elapsed () const  [inline]
```

Get the elapsed time between calls to this object's **start()** (p. 812) and **stop()** (p. 812) methods.

Returns

Elapsed time converted to the integral units requested, which may experience loss of precision.

Exceptions

Error::StrategyError *(p. 789)*	Propagated from **elapsedTimePoint()** (p. 812).

Note

Values returned from this method are limited in their precision by the resolution of BE_CLOCK_TYPE. For example, if the clock's native resolution is in nanoseconds, requesting a picoseconds representation returns 1000 times the nanosecond value, not the true value in picoseconds.

Returned values are limited by the semantics of C++'s duration type, which reports only **whole** units. For example, if a representation of hours was requested, 0 would be returned for durations of less than 3600s, and 1 would be returned for durations of 3600s through and including 7199s. For floating point approximations of representations, use **elapsedTimePoint()** (p. 812) to obtain a std::chrono::duration that uses a floating point type to store the count.

H.151.4.2 elapsedStr()

```
template<typename Duration >
std::string BiometricEvaluation::Time::Timer::elapsedStr (
            bool displayUnits = false) const  [inline]
```

Convenience method for printing elapsed time as a string.

Parameters

displayUnits	Append the elapsed time units.

Returns

String representing the elapsed time.

Exceptions

Error::StrategyError *(p. 789)*	Propagated from **elapsed**<**Duration**>() (p. 811) or **units**<**Duration**>() (p. 813).

Note

See the important note in **elapsed()** (p. 811).

H.151.4.3 elapsedTimePoint()

```
std::common_type_t< BE_CLOCK_TYPE::time_point::duration, BE_CLOCK_TYPE::time_point::duration > Biometric↩
Evaluation::Time::Timer::elapsedTimePoint () const
```

Get the elapsed time between calls to this object's **start()** (p. 812) and **stop()** (p. 812) methods.

Returns

Elapsed time.

Exceptions

Error::StrategyError *(p. 789)*	This object is currently timing an operation or an error occurred when obtaining timing informati

@seealso **elapsed()** (p. 811)

Note

This method may be useful for obtaining floating point representations of durations. For example, `std↩ ::chrono::duration<double, std::milli>( std::chrono::microseconds(1001599)).count()` would return a `double` with the value 1001.599000, the millisecond floating point representation of 1001599 microseconds.

H.151.4.4 start()

```
void BiometricEvaluation::Time::Timer::start ()
```

Start tracking time.

Exceptions

Error::StrategyError *(p. 789)*	This object is currently timing an operation or an error occurred when obtaining timing informati

H.151.4.5 stop()

```
void BiometricEvaluation::Time::Timer::stop ()
```

Stop tracking time.

Exceptions

Error::StrategyError *(p. 789)*	This object is not currently timing an operation or an error occurred when obtaining timing inforr

H.151.4.6 time()

```
Timer & BiometricEvaluation::Time::Timer::time (
            const std::function< void()> & func)
```

Record the runtime of a function.

Parameters

func	Function to time.

Returns

Reference to this class.

Exceptions

Error::StrategyError *(p. 789)*	Propagated from **start()** (p. 812) or **stop()** (p. 812), and/or func is nullptr.

H.151.4.7 units()

```
template<typename Duration >
static std::string BiometricEvaluation::Time::Timer::units ()  [inline], [static]
```

Returns

Unit label for a particular duration.

Exceptions

Error::StrategyError *(p. 789)*	Unrecognized duration encountered and units cannot be determined.

H.152 BiometricEvaluation::Device::TLV Class Reference

A class to represent a Tag-Length-Value (**TLV** (p. 813)) data structure as described in the ISO 7816-4 integrated circuit card standard.

`#include <be_device_tlv.h>`

Public Member Functions

- **TLV** ()

  Construct an empty Tag-Length-Value object that can be filled with setter methods.
- **TLV** (const **Memory::uint8Array** &buf)

  Construct a Tag-Length-Value object from the given buffer.
- **TLV** (**Memory::IndexedBuffer** &ibuf)

  *Construct a single **TLV** (p. 813) from the indexed buffer.*
- **TLV** (const std::string &filename)

  Construct a Tag-Length-Value object from the given file name.
- void **setTag** (const **Memory::uint8Array** &tag)

  Set the encoded tag value.
- const **Memory::uint8Array getTag** () const

Obtain the encoded tag value.

- uint32_t **getTagNum** () const
- uint8_t **getTagClass** () const
- bool **isPrimitive** () const
- void **setPrimitive** (const **Memory::uint8Array** &value)

  *Set the primitive data associated with this **TLV** (p. 813).*
- **Memory::uint8Array** **getPrimitive** () const

  *Obtain the primitive data associated with this **TLV** (p. 813).*
- void **addChild** (const **TLV** &tlv)
- std::vector< **TLV** > **getChildren** () const
- **Memory::uint8Array** **getRawTLV** () const

  *Obtain the **TLV** (p. 813) as an array of 8-bit values.*

Static Public Member Functions

- static std::string **stringFromTLV** (const **TLV** &tlv, const int tabCount)

  *Class utility function to print the contents of a **TLV** (p. 813) into a string object, in readable format.*

H.152.1 Detailed Description

A class to represent a Tag-Length-Value (**TLV** (p. 813)) data structure as described in the ISO 7816-4 integrated circuit card standard.

A **TLV** (p. 813) is composed of tag and length fields, then a value field that may be another **TLV** (p. 813) (a child), or data of another format, represented as the primitive object in this class.

H.152.2 Constructor & Destructor Documentation

H.152.2.1 TLV() [1/4]

```
BiometricEvaluation::Device::TLV::TLV ()
```

Construct an empty Tag-Length-Value object that can be filled with setter methods.

Empty **TLV** (p. 813) objects are primitive.

H.152.2.2 TLV() [2/4]

```
BiometricEvaluation::Device::TLV::TLV (
            const  Memory::uint8Array & buf)
```

Construct a Tag-Length-Value object from the given buffer.

Exceptions

Error::DataError *(p. 390)*	The data in the buffer is not conforming.

H.152.2.3 TLV() [3/4]

```
BiometricEvaluation::Device::TLV::TLV (
            Memory::IndexedBuffer & ibuf)
```

Construct a single **TLV** (p. 813) from the indexed buffer.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) parsing the data in the buffer.

H.152.2.4 TLV() [4/4]

```
BiometricEvaluation::Device::TLV::TLV (
            const std::string & filename)
```

Construct a Tag-Length-Value object from the given file name.

Exceptions

Error::DataError *(p. 390)*	The data in the file is not conformance.

H.152.3 Member Function Documentation

H.152.3.1 addChild()

```
void BiometricEvaluation::Device::TLV::addChild (
            const  TLV & tlv)
```

Add a child **TLV** (p. 813).

Parameters

tlv	The **TLV** (p. 813) to be added as a child of this **TLV** (p. 813).

Exceptions

Error::DataError *(p. 390)*	The **TLV** (p. 813) is primitive.

H.152.3.2 getChildren()

```
std::vector<  TLV > BiometricEvaluation::Device::TLV::getChildren () const
```

Get copies of the child TLVs.

Returns

A vector of child TLVs.

Exceptions

Error::DataError *(p. 390)*	The **TLV** (p. 813) is primitive.

H.152.3.3 getPrimitive()

`Memory::uint8Array BiometricEvaluation::Device::TLV::getPrimitive () const`

Obtain the primitive data associated with this **TLV** (p. 813).

Exceptions

Error::DataError *(p. 390)*	The **TLV** (p. 813) is of the constructed form.

See also

getChildren (p. 815).

H.152.3.4 getRawTLV()

`Memory::uint8Array BiometricEvaluation::Device::TLV::getRawTLV () const`

Obtain the **TLV** (p. 813) as an array of 8-bit values.

The array can be sent to a device that accepts TLV-encoded objects, typically wrapped in device command structures.

Returns

The **TLV** (p. 813) as an array.

H.152.3.5 getTagClass()

`uint8_t BiometricEvaluation::Device::TLV::getTagClass () const`

Get the decoded tag class.

Returns

The tag class.

H.152.3.6 getTagNum()

`uint32_t BiometricEvaluation::Device::TLV::getTagNum () const`

Get the decoded tag number.

Returns

The tag number.

H.152.3.7 isPrimitive()

`bool BiometricEvaluation::Device::TLV::isPrimitive () const`

Obtain the type of **TLV** (p. 813): primitive/constructed.

Returns

True if is a primitive **TLV** (p. 813), false otherwise.

H.152.3.8 setPrimitive()

```
void BiometricEvaluation::Device::TLV::setPrimitive (
            const  Memory::uint8Array & value)
```

Set the primitive data associated with this **TLV** (p. 813).

The primitive data is added as the value data item.

Exceptions

Error::DataError *(p. 390)*	The **TLV** (p. 813) is already of the constructed form, meaning that there are **TLV** (p. 813) children se

H.152.3.9 setTag()

```
void BiometricEvaluation::Device::TLV::setTag (
            const  Memory::uint8Array & tag)
```

Set the encoded tag value.

This function will cause a recalculation of the decoded tag number, class and primitive indicators.

Exceptions

Error::DataError *(p. 390)*	The primitive indicator conflicts with the presence of children TLVs, or presence of primitive d
Error::ParameterError *(p. 655)*	The length of the buffer is larger than the maximum tag length.

H.152.3.10 stringFromTLV()

```
static std::string BiometricEvaluation::Device::TLV::stringFromTLV (
            const  TLV & tlv,
            const int tabCount)  [static]
```

Class utility function to print the contents of a **TLV** (p. 813) into a string object, in readable format.

Parameters

tlv	The **TLV** (p. 813) to print.
tabCount	The number of tab characters to insert before each line of the output.

H.153 BiometricEvaluation::Memory::unique_if< T > Struct Template Reference

Define a type that is visible when T is not an array.

`#include <be_memory.h>`

Public Types

- using **unique_single** = std::unique_ptr<T>

H.153.1 Detailed Description

template<class T>
struct BiometricEvaluation::Memory::unique_if< T >

Define a type that is visible when T is not an array.

Note

Coming in C++14. This implementation is taken from the LLVM implementation.

H.153.2 Member Typedef Documentation

H.153.2.1 unique_single

```
template<class T >
using  BiometricEvaluation::Memory::unique_if< T >::unique_single = std::unique_ptr<T>
```

Type to use when T is not an array.

H.154 BiometricEvaluation::Memory::unique_if< T[]> Struct Template Reference

Define a type that is visible when T is an unknown-bound array.

`#include <be_memory.h>`

Public Types

- using **unique_array_unknown_bound** = std::unique_ptr<T[ ]>

H.154.1 Detailed Description

template<class T>
struct BiometricEvaluation::Memory::unique_if< T[ ]>

Define a type that is visible when T is an unknown-bound array.

Note

Coming in C++14. This implementation is taken from the LLVM implementation.

H.154.2 Member Typedef Documentation

H.154.2.1 unique_array_unknown_bound

```
template<class T >
using BiometricEvaluation::Memory::unique_if< T[]>::unique_array_unknown_bound = std::unique_ptr<T[]>
```

Type to use when T is unknown-bound array.

H.155 BiometricEvaluation::Memory::unique_if< T[S]> Struct Template Reference

Define a type that is visible when T is an known-bound array.

`#include <be_memory.h>`

Public Types

- using **unique_array_known_bound** = void

H.155.1 Detailed Description

template<class T, size_t S>
struct BiometricEvaluation::Memory::unique_if< T[S]>

Define a type that is visible when T is an known-bound array.

Note

Coming in C++14. This implementation is taken from the LLVM implementation.

H.155.2 Member Typedef Documentation

H.155.2.1 unique_array_known_bound

```
template<class T , size_t S>
using BiometricEvaluation::Memory::unique_if< T[S]>::unique_array_known_bound = void
```

Type to use when T is known-bound array.

H.156 BiometricEvaluation::View::View Class Reference

A class to represent single biometric element view.

`#include <be_view_view.h>`

Inheritance diagram for BiometricEvaluation::View::View:

Public Member Functions

- std::shared_ptr< **Image::Image** > **getImage** () const

  Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)
- **Image::Size getImageSize** () const

  Obtain the image size.
- **Image::Resolution getImageResolution** () const

  Obtain the image resolution.
- uint32_t **getImageColorDepth** () const

  Obtain the image color depth in bits-per-pixel.
- **Image::CompressionAlgorithm getCompressionAlgorithm** () const

  Obtain the compression algorithm used on the image.
- **Image::Resolution getScanResolution** () const

  Obtain the image scan resolution.

Protected Member Functions

- void **setImageSize** (const **BiometricEvaluation::Image::Size** &imageSize)

  Mutator for the image size.
- void **setImageColorDepth** (uint32_t imageColorDepth)

  Mutator for the image color depth.
- void **setImageResolution** (const **BiometricEvaluation::Image::Resolution** &imageResolution)

  Mutator for the image resolution.
- void **setScanResolution** (const **BiometricEvaluation::Image::Resolution** &scanResolution)

  Mutator for the image scan resolution.
- void **setImageData** (const **BiometricEvaluation::Memory::uint8Array** &imageData)

  Mutator for the image data.
- void **setCompressionAlgorithm** (const **Image::CompressionAlgorithm** &ca)

  Mutator for the compression algorithm.

H.156.1 Detailed Description

A class to represent single biometric element view.

Included in a view is the biometric image and any derived information, such as minutiae points.

H.156.2 Member Function Documentation

H.156.2.1 getCompressionAlgorithm()

Image::CompressionAlgorithm BiometricEvaluation::View::View::getCompressionAlgorithm () const

Obtain the compression algorithm used on the image.

This value is as present in the biometric record, and not obtained from the image data itself.

Returns

The compression algorithm.

H.156.2.2 getImage()

`std::shared_ptr< Image::Image > BiometricEvaluation::View::View::getImage () const`

Obtain the image used for the biometric view in the format contained in the record (JPEG, etc.)

Not all views will have an image, however the derived information, such as minutiae, may be present.

Returns

The image data.

H.156.2.3 getImageColorDepth()

`uint32_t BiometricEvaluation::View::View::getImageColorDepth () const`

Obtain the image color depth in bits-per-pixel.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image depth must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

Returns

The image depth.

H.156.2.4 getImageResolution()

`Image::Resolution BiometricEvaluation::View::View::getImageResolution () const`

Obtain the image resolution.

Image (p. 128) resolution is taken from the biometric record, and not from the image data.

Returns

The scan resolution.

Note

In some cases, the resolution may be the components of the pixel ratio, and applications must check the **Image::Resolution::Units** (p. 738) field for value NA.

H.156.2.5 getImageSize()

`Image::Size BiometricEvaluation::View::View::getImageSize () const`

Obtain the image size.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image size must be equal, but applications can check for inconsistencies. In the case of raw images, however, the value obtained with this method must be accepted as correct.

Returns

The image size.

H.156.2.6 getScanResolution()

`Image::Resolution BiometricEvaluation::View::View::getScanResolution () const`

Obtain the image scan resolution.

This value is as present in the biometric record, and not in the image data itself. Normally, this value and the actual image resolution must be equal, but applications can check for inconsistencies.

Returns

The scan resolution.

Note

In some cases, the resolution may be the components of the pixel ratio, and applications must check the **Image::Resolution::Units** (p. 738) field for value NA.

H.156.2.7 setImageColorDepth()

```
void BiometricEvaluation::View::View::setImageColorDepth (
            uint32_t imageColorDepth)  [protected]
```

Mutator for the image color depth.

Parameters

in	*imageColorDepth*	The image color depth.

H.156.2.8 setImageData()

```
void BiometricEvaluation::View::View::setImageData (
            const  BiometricEvaluation::Memory::uint8Array & imageData)  [protected]
```

Mutator for the image data.

Parameters

in	*imageData*	The image data object.

H.156.2.9 setImageResolution()

```
void BiometricEvaluation::View::View::setImageResolution (
            const  BiometricEvaluation::Image::Resolution & imageResolution)  [protected]
```

Mutator for the image resolution.

Parameters

in	*imageResolution*	The image resolution object.

H.156.2.10 setImageSize()

```
void BiometricEvaluation::View::View::setImageSize (
            const  BiometricEvaluation::Image::Size & imageSize)  [protected]
```

Mutator for the image size.

Parameters

in	*imageSize*	The image size object.

H.156.2.11 setScanResolution()

```
void BiometricEvaluation::View::View::setScanResolution (
            const  BiometricEvaluation::Image::Resolution & scanResolution)  [protected]
```

Mutator for the image scan resolution.

Parameters

in	*scanResolution*	The image scan resolution object.

H.157 BiometricEvaluation::Time::Watchdog Class Reference

A **Watchdog** (p. 823) object can be used by applications to limit the amount of processing time taken by a block of code.

`#include <be_time_watchdog.h>`

Public Member Functions

- **Watchdog** (const uint8_t type)
- uint64_t **getInterval** () const noexcept

  Obtain the timer interval.

- void **setInterval** (uint64_t interval)
- void **start** ()
- void **stop** ()
- bool **expired** ()
- void **setCanSigJump** ()
- void **clearCanSigJump** ()
- void **setExpired** ()
- void **clearExpired** ()
- void **setEnabled** (const bool enabled)
- bool **isEnabled** () const

Static Public Attributes

- static const uint8_t **PROCESSTIME** = 0
- static const uint8_t **REALTIME** = 1
- static bool **_canSigJump**
- static sigjmp_buf **_sigJumpBuf**

H.157.1 Detailed Description

A **Watchdog** (p. 823) object can be used by applications to limit the amount of processing time taken by a block of code.

A **Watchdog** (p. 823) 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** (p. 823) builds on the POSIX setitimer(2) call. **Timer** (p. 809) intervals are in terms of process virtual time or real time, based on how the object is constructed.

Most applications will not directly invoke the methods of the WatchDog class, instead using the BEGIN_↩ WATCHDOG_BLOCK() and END_WATCHDOG_BLOCK() macros. Applications should not install their own signal handlers, but use the SignalManager class instead.

The BEGIN_WATCHDOG_BLOCK() macro sets up the jump block and tells the **Watchdog** (p. 823) object to start handling the alarm signal. Applications must call **setInterval()** (p. 827) before invoking the BEGIN_↩ WATCHDOG_BLOCK() macro.

The END_WATCHDOG_BLOCK() macro disables the watchdog timer, but doesn't affect the assigned interval value. Applications can set the interval once and use the block macros repeatedly. Failure to call **set**↩ **Interval()** (p. 827) results in an effectively disabled timer, as does setting the interval to 0.

The ABORT_WATCHDOG() macro also disables the watchdog timer but does not create the code point destination for the jump point. This macro should be used to disable a **Watchdog** (p. 823) object when the application is no longer interested in the timeout condition.

Attention

The BEGIN_WATCHDOG_BLOCK() macro must be paired with either the END_WATCHDOG_BLOCK() macro or ABORT_WATCHDOG_BLOCK() macro. Failure to do so may result in undefined behavior as a running **Watchdog** (p. 823) timer may expire, forcing a jump into an incompletely initialized function.

Note

Process (p. 170) 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** (p. 823) class. Therefore, applications should not call sleep(3) inside the **Watchdog** (p. 823) block; behavior is undefined in that case, but usually results in cancellation of the **Watchdog** (p. 823) timer.

The **setCanSigJump()** (p. 826), **clearCanSigJump()** (p. 825), **setExpired()** (p. 826) and **clearExpired()** (p. 825) 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 (p. 759)

H.157.2 Constructor & Destructor Documentation

H.157.2.1 Watchdog()

```
BiometricEvaluation::Time::Watchdog::Watchdog (
            const uint8_t type)
```

Construct a new **Watchdog** (p. 823) object.

Parameters

in	*type*	The type of timer, Process↩ Time or Real↩ Time.

Exceptions

Error::NotImplemented (p. 636)	The type of watchdog requested is not implemented.
Error::ParameterError (p. 655)	The type is invalid.

Warning

Watchdog::PROCESSTIME (p. 827) is not supported under Cygwin.

H.157.3 Member Function Documentation

H.157.3.1 clearCanSigJump()

```
void BiometricEvaluation::Time::Watchdog::clearCanSigJump ()
```

Clears the flag for the **Watchdog** (p. 823) object to indicate that the signal jump block is no longer valid.

H.157.3.2 clearExpired()

```
void BiometricEvaluation::Time::Watchdog::clearExpired ()
```

Clear the flag indicating the timer expired.

H.157.3.3 expired()

```
bool BiometricEvaluation::Time::Watchdog::expired ()
```

Indicate whether the watchdog timer expired.

Returns

true if the timer expired, false otherwise.

H.157.3.4 getInterval()

```
uint64_t BiometricEvaluation::Time::Watchdog::getInterval () const  [noexcept]
```

Obtain the timer interval.

Returns

Current timer interval.

H.157.3.5 isEnabled()

```
bool BiometricEvaluation::Time::Watchdog::isEnabled () const
```

Check the enabled status of the timer.

H.157.3.6 setCanSigJump()

```
void BiometricEvaluation::Time::Watchdog::setCanSigJump ()
```

Indicate that the signal handler can jump into the application code after handling the signal.

H.157.3.7 setEnabled()

```
void BiometricEvaluation::Time::Watchdog::setEnabled (
            const bool enabled)
```

Enable or disable the timer.

Parameters

enabled	`true` if enabled, `false` otherwise.

Note

This enables easier debugging without changing sourcecode to remove **Watchdog** (p. 823) blocks.

H.157.3.8 setExpired()

```
void BiometricEvaluation::Time::Watchdog::setExpired ()
```

Set a flag to indicate the timer expired.

H.157.3.9 setInterval()

```
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** (p. 809) 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.

H.157.3.10 start()

```
void BiometricEvaluation::Time::Watchdog::start ()
```

Start a watchdog timer.

Exceptions

Error::StrategyError *(p. 789)*	Could not register the signal handler, or could not create the timer.

H.157.3.11 stop()

```
void BiometricEvaluation::Time::Watchdog::stop ()
```

Stop a watchdog timer.

Exceptions

Error::StrategyError *(p. 789)*	Could not clear the timer.

H.157.4 Member Data Documentation

H.157.4.1 PROCESSTIME

```
const uint8_t BiometricEvaluation::Time::Watchdog::PROCESSTIME = 0  [static]
```

A **Watchdog** (p. 823) based on process time.

H.157.4.2 REALTIME

```
const uint8_t BiometricEvaluation::Time::Watchdog::REALTIME = 1  [static]
```

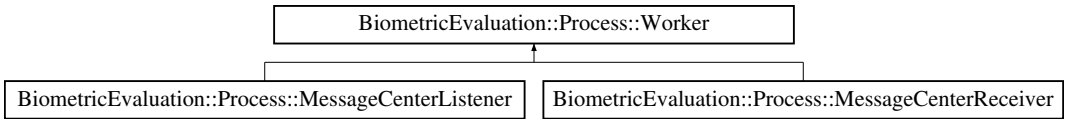
A **Watchdog** (p. 823) based on real (wall clock) time.

H.158 BiometricEvaluation::Process::Worker Class Reference

An abstraction of an instance that performs work on given data.

`#include <be_process_worker.h>`

Inheritance diagram for BiometricEvaluation::Process::Worker:



Public Member Functions

- virtual int32_t **workerMain** ()=0

  The method that will get called to start execution by a ProcessManager.
- std::shared_ptr< void > **getParameter** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828).*
- double **getParameterAsDouble** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828) as a double.*
- int64_t **getParameterAsInteger** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828) as an integer.*
- std::string **getParameterAsString** (const std::string &name)

  *Obtain a parameter passed to this **Worker** (p. 828) as a string.*
- void **setParameter** (const std::string &name, std::shared_ptr< void > argument)

  *Pass a parameter to this **Worker** (p. 828).*
- virtual void **stop** () final

  *Tell this **Worker** (p. 828) to return ASAP.*
- void **closeWorkerPipeEnds** ()

  *Perform initialization for communication from **Worker** (p. 828) to **Manager** (p. 596).*
- void **closeManagerPipeEnds** ()

  *Perform initialization for communication from **Manager** (p. 596) to **Worker** (p. 828).*
- int **getSendingPipe** () const

  *Obtain the pipe used to send messages to this **Worker** (p. 828).*
- int **getReceivingPipe** () const

  *Obtain the pipe used to receive messages to this **Worker** (p. 828).*
- void **sendMessageToManager** (const **Memory::uint8Array** &message)

  *Send a message to the **Manager** (p. 596).*
- void **receiveMessageFromManager** (**Memory::uint8Array** &message)

  *Receive a message from the **Manager** (p. 596).*
- void **_initCommunication** ()

  Perform general communication initialization from Constructor.
- virtual ~**Worker** ()

  ***Worker** (p. 828) destructor.*

Protected Member Functions

- **Worker** ()

  Worker *(p. 828) constructor.*
- virtual bool **stopRequested** () const final

  Determine if the parent has requested this child to exit.
- bool **waitForMessage** (int numSeconds=-1) const

  Block while waiting for a message from the ***Manager*** *(p. 596).*

H.158.1 Detailed Description

An abstraction of an instance that performs work on given data.

H.158.2 Member Function Documentation

H.158.2.1 _initCommunication()

`void BiometricEvaluation::Process::Worker::_initCommunication ()`

Perform general communication initialization from Constructor.

Exceptions

Error::StrategyError *(p. 789)*	**Error** (p. 112) in initialization.

H.158.2.2 closeManagerPipeEnds()

`void BiometricEvaluation::Process::Worker::closeManagerPipeEnds ()`

Perform initialization for communication from **Manager** (p. 596) to **Worker** (p. 828).

Note

Behavior is undefined if called by a non-Worker.

Exceptions

Error::StrategyError *(p. 789)*	Communications not enabled.

H.158.2.3 closeWorkerPipeEnds()

`void BiometricEvaluation::Process::Worker::closeWorkerPipeEnds ()`

Perform initialization for communication from **Worker** (p. 828) to **Manager** (p. 596).

Note

Behavior is undefined if called by a non-Manager.

Exceptions

Error::StrategyError *(p. 789)*	Communications not enabled.

H.158.2.4 getParameter()

```
std::shared_ptr< void > BiometricEvaluation::Process::Worker::getParameter (
            const std::string & name)
```

Obtain a parameter passed to this **Worker** (p. 828).

Parameters

name	The parameter name to retrieve.

Returns

shared_ptr to the parameter argument.

Exceptions

std::out_of_range	name was not set.

H.158.2.5 getParameterAsDouble()

```
double BiometricEvaluation::Process::Worker::getParameterAsDouble (
            const std::string & name)
```

Obtain a parameter passed to this **Worker** (p. 828) as a double.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as a double.

Exceptions

std::out_of_range	name was not set.

H.158.2.6 getParameterAsInteger()

```
int64_t BiometricEvaluation::Process::Worker::getParameterAsInteger (
            const std::string & name)
```

Obtain a parameter passed to this **Worker** (p. 828) as an integer.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as an integer.

Exceptions

std::out_of_range	name was not set.

H.158.2.7 getParameterAsString()

```
std::string BiometricEvaluation::Process::Worker::getParameterAsString (
            const std::string & name)
```

Obtain a parameter passed to this **Worker** (p. 828) as a string.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as a string.

Exceptions

std::out_of_range	name was not set.

H.158.2.8 getReceivingPipe()

```
int BiometricEvaluation::Process::Worker::getReceivingPipe () const
```

Obtain the pipe used to receive messages to this **Worker** (p. 828).

Returns

Receiving pipe.

Exceptions

Error::ObjectDoesNotExist (p. 637)	**Worker** (p. 828) exiting soon, communication disabled.
Error::StrategyError (p. 789)	Communications not enabled.

H.158.2.9 getSendingPipe()

```
int BiometricEvaluation::Process::Worker::getSendingPipe () const
```

Obtain the pipe used to send messages to this **Worker** (p. 828).

Returns

Sending pipe.

Exceptions

Error::ObjectDoesNotExist (p. 637)	**Worker** (p. 828) exiting soon, communication disabled.
Error::StrategyError (p. 789)	Communications not enabled.

H.158.2.10 receiveMessageFromManager()

```
void BiometricEvaluation::Process::Worker::receiveMessageFromManager (
            Memory::uint8Array & message)
```

Receive a message from the **Manager** (p. 596).

Parameters

out	*message*	Buffer to store the received message.

Exceptions

Error::ObjectDoesNotExist (p. 637)	Widowed pipe.
Error::StrategyError (p. 789)	Communications not enabled.

See also

waitForMessage (p. 833)

H.158.2.11 sendMessageToManager()

```
void BiometricEvaluation::Process::Worker::sendMessageToManager (
            const Memory::uint8Array & message)
```

Send a message to the **Manager** (p. 596).

Parameters

in	*message*	Message to send.

Exceptions

Error::ObjectDoesNotExist (p. 637)	Widowed pipe.
Error::StrategyError (p. 789)	Communications not enabled.

H.158.2.12 setParameter()

```
void BiometricEvaluation::Process::Worker::setParameter (
            const std::string & name,
            std::shared_ptr< void > argument)
```

Pass a parameter to this **Worker** (p. 828).

Parameters

name	A unique identifier for this parameter
argument	A shared_ptr to the object to store.

H.158.2.13 stopRequested()

```
virtual bool BiometricEvaluation::Process::Worker::stopRequested () const  [final], [protected],
[virtual]
```

Determine if the parent has requested this child to exit.

Returns

Whether or not this child should exit.

H.158.2.14 waitForMessage()

```
bool BiometricEvaluation::Process::Worker::waitForMessage (
            int numSeconds = -1) const  [protected]
```

Block while waiting for a message from the **Manager** (p. 596).

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.

H.158.2.15 workerMain()

```
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** (p. 447) object, the implementation of **Process::Worker::workerMain()** (p. 834) should release all resources prior to returning.

Any exceptions thrown by this method will cause the worker to exit with a return status of EXIT_↩ FAILURE. The type and contents of the exception is not maintained.

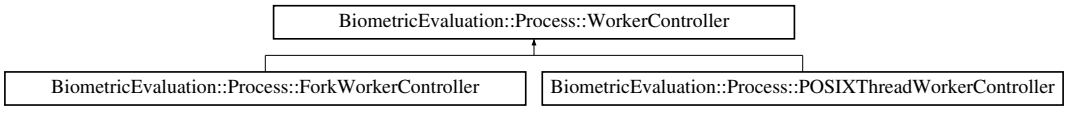
Implemented in **BiometricEvaluation::Process::MessageCenterListener** (p. 612), and **Biometric↩ Evaluation::Process::MessageCenterReceiver** (p. 614).

H.159 BiometricEvaluation::Process::WorkerController Class Reference

Wrapper of a **Worker** (p. 828) returned from a **Process::Manager** (p. 596).

`#include <be_process_workercontroller.h>`

Inheritance diagram for BiometricEvaluation::Process::WorkerController:



Public Member Functions

- **WorkerController** (std::shared_ptr< **Worker** > worker)
- virtual void **sendMessageToWorker** (const **Memory::uint8Array** &message)

  *Send a message to the **Worker** (p. 828) contained within this **WorkerController** (p. 834).*
- virtual void **setParameter** (const std::string &name, std::shared_ptr< void > argument)

  *Set the parameter to be passed to the **Worker** (p. 828).*
- virtual void **setParameterFromDouble** (const std::string &name, double argument)

  *Set a double parameter to be passed to the **Worker** (p. 828).*
- virtual void **setParameterFromInteger** (const std::string &name, int64_t argument)

  *Set an integer parameter to be passed to the **Worker** (p. 828).*
- virtual void **setParameterFromString** (const std::string &name, const std::string &argument)

  *Set a string parameter to be passed to the **Worker** (p. 828).*
- virtual void **reset** ()

  *Reuse the **Worker** (p. 828).*
- virtual bool **isWorking** () const =0

  *Obtain whether or not **Worker** (p. 828) is working.*
- virtual bool **everWorked** () const =0

  *Obtain whether or not this **Worker** (p. 828) has ever worked.*
- bool **finishedWorking** () const

  *Obtain whether or not this **Worker** (p. 828) has both started and finished its task.*
- std::shared_ptr< **Worker** > **getWorker** () const

  *Obtain the **Worker** (p. 828) instance being wrapped.*
- virtual int32_t **getExitStatus** () const final

  *Obtain the exit status of the wrapped **Worker** (p. 828).*
- virtual ~**WorkerController** ()

  ***WorkerController** (p. 834) destructor.*

Protected Attributes

- std::shared_ptr< **Worker** > **_worker**
- bool **_rvSet**
- int32_t **_rv**

H.159.1 Detailed Description

Wrapper of a **Worker** (p. 828) returned from a **Process::Manager** (p. 596).

H.159.2 Constructor & Destructor Documentation

H.159.2.1 WorkerController()

```
BiometricEvaluation::Process::WorkerController::WorkerController (
            std::shared_ptr< Worker > worker)
```

WorkerController (p. 834) constructor.

Parameters

worker	The **Worker** (p. 828) instance to wrap.

H.159.3 Member Function Documentation

H.159.3.1 everWorked()

```
virtual bool BiometricEvaluation::Process::WorkerController::everWorked () const  [pure virtual]
```

Obtain whether or not this **Worker** (p. 828) has ever worked.

Returns

true the **Worker** (p. 828) has ever or is currently working, false otherwise.

Note

reset() (p. 837) will change the result of this method.

Implemented in **BiometricEvaluation::Process::ForkWorkerController** (p. 459), and **BiometricEvaluation::Process::POSIXThreadWorkerController** (p. 673).

H.159.3.2 finishedWorking()

```
bool BiometricEvaluation::Process::WorkerController::finishedWorking () const  [inline]
```

Obtain whether or not this **Worker** (p. 828) has both started and finished its task.

Returns

true if the **Worker** (p. 828) has both started and finished performing its task, false otherwise.

Note

reset() (p. 837) will change the result of this method.

H.159.3.3 getExitStatus()

```
virtual int32_t BiometricEvaluation::Process::WorkerController::getExitStatus () const  [final],
[virtual]
```

Obtain the exit status of the wrapped **Worker** (p. 828).

Returns

Exit status of the wrapped **Worker** (p. 828).

Exceptions

Error::ObjectDoesNotExist (p. 637)	Exit status not set.
Error::StrategyError (p. 789)	Exit status not set (e.g., **Worker** (p. 828) has not been started or **Worker** (p. 828) has not fi

H.159.3.4 getWorker()

`std::shared_ptr< Worker > BiometricEvaluation::Process::WorkerController::getWorker () const`

Obtain the **Worker** (p. 828) instance being wrapped.

Returns

Worker (p. 828) instance.

H.159.3.5 isWorking()

`virtual bool BiometricEvaluation::Process::WorkerController::isWorking () const [pure virtual]`

Obtain whether or not **Worker** (p. 828) is working.

Returns

Whether or not the **Worker** (p. 828) is working.

Implemented in **BiometricEvaluation::Process::ForkWorkerController** (p. 459), and **BiometricEvaluation::Process::POSIXThreadWorkerController** (p. 673).

H.159.3.6 reset()

`virtual void BiometricEvaluation::Process::WorkerController::reset () [virtual]`

Reuse the **Worker** (p. 828).

Exceptions

Error::ObjectExists (p. 637)	The previously started **Worker** (p. 828) is still running.

Reimplemented in **BiometricEvaluation::Process::ForkWorkerController** (p. 459), and **BiometricEvaluation::Process::POSIXThreadWorkerController** (p. 673).

H.159.3.7 sendMessageToWorker()

```
virtual void BiometricEvaluation::Process::WorkerController::sendMessageToWorker (
            const Memory::uint8Array & message) [virtual]
```

Send a message to the **Worker** (p. 828) contained within this **WorkerController** (p. 834).

Parameters

message	Message to send to the **Worker** (p. 828).

Exceptions

Error::ObjectDoesNotExist *(p. 637)*	**Worker** (p. 828) receive pipe is closed (**Worker** (p. 828) object likely destroyed).
Error::StrategyError *(p. 789)*	Message sending failed.

H.159.3.8 setParameter()

```
virtual void BiometricEvaluation::Process::WorkerController::setParameter (
            const std::string & name,
            std::shared_ptr< void > argument)  [virtual]
```

Set the parameter to be passed to the **Worker** (p. 828).

Parameters

in	*name*	The name representing the argument in the **Worker** (p. 828).
in	*argument*	The argument to be passed to the **Worker** (p. 828).

Note

Subsequent calls to **setParameter()** (p. 838) with the same name will overwrite any exiting argument.

H.159.3.9 setParameterFromDouble()

```
virtual void BiometricEvaluation::Process::WorkerController::setParameterFromDouble (
            const std::string & name,
            double argument)  [virtual]
```

Set a double parameter to be passed to the **Worker** (p. 828).

Parameters

in	*name*	The name representing the argument in the **Worker** (p. 828).
in	*argument*	The double to be passed to the **Worker** (p. 828).

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

H.159.3.10 setParameterFromInteger()

```
virtual void BiometricEvaluation::Process::WorkerController::setParameterFromInteger (
            const std::string & name,
            int64_t argument)  [virtual]
```

Set an integer parameter to be passed to the **Worker** (p. 828).

Parameters

in	*name*	The name representing the argument in the **Worker** (p. 828).
in	*argument*	The integer to be passed to the **Worker** (p. 828).

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

H.159.3.11 setParameterFromString()

```
virtual void BiometricEvaluation::Process::WorkerController::setParameterFromString (
            const std::string & name,
            const std::string & argument)  [virtual]
```

Set a string parameter to be passed to the **Worker** (p. 828).

Parameters

in	*name*	The name representing the argument in the **Worker** (p. 828).
in	*argument*	The string to be passed to the **Worker** (p. 828).

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

H.159.4 Member Data Documentation

H.159.4.1 _rv

```
int32_t BiometricEvaluation::Process::WorkerController::_rv  [protected]
```

Exit status from _worker.workerMain()

H.159.4.2 _rvSet

```
bool BiometricEvaluation::Process::WorkerController::_rvSet  [protected]
```

Whether or not _rv contains a true value.

H.159.4.3 _worker

```
std::shared_ptr< Worker> BiometricEvaluation::Process::WorkerController::_worker  [protected]
```

The **Worker** (p. 828) instance that is running in this child

H.160 BiometricEvaluation::MPI::WorkPackage Class Reference

A class to represent a piece of work to be acted upon by a processor.

`#include <be_mpi_workpackage.h>`

Public Member Functions

- **WorkPackage** ()

  Construct an empty work package.
- **WorkPackage** (const **Memory::uint8Array** &data)

  Construct a work package with some data.
- void **getData** (**Memory::uint8Array** &data) const

  Obtain the package data in raw form.
- void **setData** (const **Memory::uint8Array** &data)

  Set the package data from raw data.
- uint64_t **getSize** () const

  Obtain the size of the package data.
- uint64_t **getNumElements** () const

  Obtain the number of elements in the package.
- void **setNumElements** (const uint64_t numElements)

  Set the number of elements in the package.

H.160.1 Detailed Description

A class to represent a piece of work to be acted upon by a processor.

The work package is an wrapper around the data to be processed, along with some ancillary information.

H.160.2 Constructor & Destructor Documentation

H.160.2.1 WorkPackage()

```
BiometricEvaluation::MPI::WorkPackage::WorkPackage (
            const  Memory::uint8Array & data)
```

Construct a work package with some data.

Parameters

in	*data*	The data that will be managed by this work package.

H.160.3 Member Function Documentation

H.160.3.1 getNumElements()

```
uint64_t BiometricEvaluation::MPI::WorkPackage::getNumElements () const
```

Obtain the number of elements in the package.

This value is determined by the application and must be set therein, otherwise 0 is returned.

Returns

The number of application defined elements in the work package.

H.160.3.2 getSize()

```
uint64_t BiometricEvaluation::MPI::WorkPackage::getSize () const
```

Obtain the size of the package data.

Returns

The size (in octets) of the raw data item.

H.160.3.3 setData()

```
void BiometricEvaluation::MPI::WorkPackage::setData (
            const  Memory::uint8Array & data)
```

Set the package data from raw data.

Parameters

in	*data*	The data copied into the work package.

H.160.3.4 setNumElements()

```
void BiometricEvaluation::MPI::WorkPackage::setNumElements (
            const uint64_t numElements)
```

Set the number of elements in the package.

Parameters

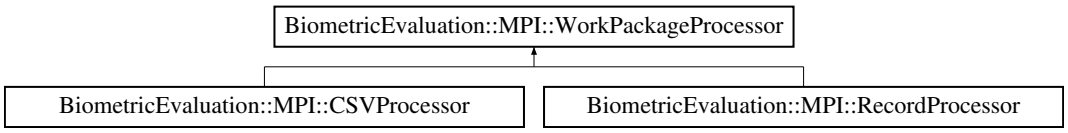
in	*numElements*	The number of appplication-defined elements in the work package.

H.161 BiometricEvaluation::MPI::WorkPackageProcessor Class Reference

Represents an object that processes the contents of a work package.

`#include <be_mpi_workpackageprocessor.h>`

Inheritance diagram for BiometricEvaluation::MPI::WorkPackageProcessor:



Public Member Functions

- virtual std::shared_ptr< **WorkPackageProcessor** > **newProcessor** (std::shared_ptr< **IO::Logsheet** > &logsheet)=0

  Obtain an object that will process work packages. This method is part of the factory personality.

- virtual void **performInitialization** (std::shared_ptr< **IO::Logsheet** > &logsheet)=0

  Initialization function to be called before work is distributed to the work package processor.

- virtual void **processWorkPackage** (**MPI::WorkPackage** &workPackage)=0

  Process (p. 170) *the data contents of the work package. This method is part of the worker personality.*

- virtual void **performShutdown** ()

  Terminiation function to be called during shut down after all work package processing is done.

- void **setLogsheet** (std::shared_ptr< **IO::Logsheet** > &logsheet)

  *Set the **IO::Logsheet*** (p. 585) *object that can be used to save message for objects of this class.*

- std::shared_ptr< **IO::Logsheet** > **getLogsheet** ()

  *Obtain the **IO::Logsheet*** (p. 585) *object that can be used to save message for objects of this class.*

H.161.1 Detailed Description

Represents an object that processes the contents of a work package.

A **WorkPackageProcessor** (p. 843) presents two personalities: One that of a worker to process work packages, and one that is a factory to return worker objects of the implementation class.

Subclasses of this class implement the functionality needed to perform an action on the work package data. The processing done by the implementation is application and data type specific.

Ultimately, the final implementation of the **WorkPackageProcessor** (p. 843) class is done in the application. Access to the Logsheet object maintained by the framework is provided by this class.

H.161.2 Member Function Documentation

H.161.2.1 getLogsheet()

```
std::shared_ptr< IO::Logsheet > BiometricEvaluation::MPI::WorkPackageProcessor::getLogsheet
()
```

Obtain the **IO::Logsheet** (p. 585) object that can be used to save message for objects of this class.

Returns

logsheet A shared pointer to the Logsheet object.

H.161.2.2 newProcessor()

```
virtual std::shared_ptr< WorkPackageProcessor > BiometricEvaluation::MPI::WorkPackageProcessor↩
::newProcessor (
            std::shared_ptr< IO::Logsheet > & logsheet) [pure virtual]
```

Obtain an object that will process work packages. This method is part of the factory personality.

Parameters

logsheet	A shared pointer to the **IO::Logsheet** (p. 585) that may be used to save messages generated by the object.

Returns

A shared pointer to the work package processor.

Note

This method should always create a non-null **WorkPackageProcessor** (p. 843). If an error occurs during construction, throw a **Error::Exception** (p. 412) with a message to be caught and logged.

Implemented in **BiometricEvaluation::MPI::CSVProcessor** (p. 384), and **BiometricEvaluation::↩ MPI::RecordProcessor** (p. 696).

H.161.2.3 performInitialization()

```
virtual void BiometricEvaluation::MPI::WorkPackageProcessor::performInitialization (
            std::shared_ptr< IO::Logsheet > & logsheet) [pure virtual]
```

Initialization function to be called before work is distributed to the work package processor.

Implementations of this class can use this function to do any processing necessary before work is given to the processor, pre-forking.

This method is part of the factory personality. All state that is to be common across all package processor objects can be initialized in this method.

Parameters

logsheet	A shared pointer to the **IO::↩ Logsheet** (p. 585) that may be used to save messages generated by the object.

Exceptions

Error::Exception *(p. 412)*	An implementation specific error occurred. The exception string will be logged by the **Framework** (

Implemented in **BiometricEvaluation::MPI::CSVProcessor** (p. 384), and **BiometricEvaluation::↩ MPI::RecordProcessor** (p. 697).

H.161.2.4 performShutdown()

```
virtual void BiometricEvaluation::MPI::WorkPackageProcessor::performShutdown ()  [virtual]
```

Terminiation function to be called during shut down after all work package processing is done.

Implementations of this class can use this function to do any processing necessary after all work is given to the processors. The default implementation does nothing.

This method is part of the factory personality. All state that is created in **performInitialization()** (p. 845) processor objects can be accessed in this method.

Exceptions

Error::Exception *(p. 412)*	An implementation specific error occurred. The exception string will be logged by the **Framework** (

H.161.2.5 processWorkPackage()

```
virtual void BiometricEvaluation::MPI::WorkPackageProcessor::processWorkPackage (
            MPI::WorkPackage & workPackage)  [pure virtual]
```

Process (p. 170) the data contents of the work package. This method is part of the worker personality.

Parameters

in	*workPackage*	The work package.

Exceptions

Error::Exception *(p. 412)*	An fatal error occurred when processing the work package; the processing responsible for this object

Implemented in **BiometricEvaluation::MPI::CSVProcessor** (p. 386), and **BiometricEvaluation::↩ MPI::RecordProcessor** (p. 699).

H.161.2.6 setLogsheet()

```
void BiometricEvaluation::MPI::WorkPackageProcessor::setLogsheet (
            std::shared_ptr< IO::Logsheet > & logsheet)
```

Set the **IO::Logsheet** (p. 585) object that can be used to save message for objects of this class.

Parameters

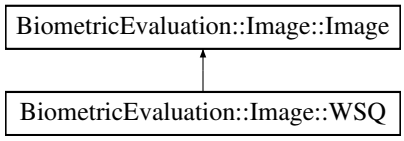
in	*logsheet*	A shared pointer to the Logsheet object.

H.162 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, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **WSQ** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)
- **Memory::uint8Array getRawData** () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::uint8Array getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.

Public Member Functions inherited from BiometricEvaluation::Image::Image

- **Image** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const **CompressionAlgorithm** compression, const bool **hasAlphaChannel**, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **Image** (const uint8_t *data, const uint64_t size, const **CompressionAlgorithm** compression, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Parent constructor for all **Image** (p. 477) classes.*
- **CompressionAlgorithm getCompressionAlgorithm** () const

  Accessor for the CompressionAlgorithm of the image.
- **Resolution getResolution** () const

  Accessor for the resolution of the image.
- **Memory::uint8Array getData** () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.
- virtual **Memory::uint8Array getRawData** (const bool removeAlphaChannelIfPresent) const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Size getDimensions** () const

  Accessor for the dimensions of the image in pixels.
- uint32_t **getColorDepth** () const

  Accessor for the color depth of the image in bits.
- uint16_t **getBitDepth** () const

  Accessor for the number of bits per color component.
- bool **hasAlphaChannel** () const

Accessor for the presence of an alpha channel.

- statusCallback_t **getStatusCallback** () const

  Get handle to status callback function.

- std::string **getIdentifier** () const

  Obtain the assigned image identifier.

Static Public Member Functions

- static bool **isWSQ** (const uint8_t *data, uint64_t size)

Static Public Member Functions inherited from BiometricEvaluation::Image::Image

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.

- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*

- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data, const std::string &identifier="", const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of a buffer of image data and create an **Image** (p. 477) object.*

- static std::shared_ptr< **Image** > **openImage** (const std::string &path, const statusCallback_t &statusCallback= **Image::defaultStatusCallback**)

  *Determine the image type of an image file and create an **Image** (p. 477) object.*

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const uint8_t *data, const uint64_t size)

  Determine the compression algorithm of a buffer of image data.

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const **Memory::uint8Array** &data)

  Determine the compression algorithm of a buffer of image data.

- static **CompressionAlgorithm** **getCompressionAlgorithm** (const std::string &path)

  Determine the compression algorithm of a file.

- static **BiometricEvaluation::Image::Raw** **getRawImage** (const std::shared_ptr< **BiometricEvaluation::Image::Image** > &image)

  *Obtain **Image::Raw** (p. 688) version of an **Image::Image** (p. 477).*

- static void **defaultStatusCallback** (const **Framework::Status** &status)

  Default handling of statuses sent from image processing libraries.

Additional Inherited Members

Public Types inherited from BiometricEvaluation::Image::Image

- using **statusCallback_t**

Protected Member Functions inherited from BiometricEvaluation::Image::Image

- void **setResolution** (const **Resolution** resolution)

  Mutator for the resolution of the image .

- void **setDimensions** (const **Size** dimensions)

  Mutator for the dimensions of the image in pixels.

- void **setColorDepth** (const uint32_t colorDepth)

Mutator for the color depth of the image in bits.

- void **setBitDepth** (const uint16_t bitDepth)

  Mutator for the number of bits per component for color components in the image, in bits.
- const uint8_t * **getDataPointer** () const
- uint64_t **getDataSize** () const
- void **setHasAlphaChannel** (const bool **hasAlphaChannel**)

  Mutator for the presence of an alpha channel.

H.162.1 Detailed Description

A WSQ-encoded image.

H.162.2 Member Function Documentation

H.162.2.1 getRawData()

`Memory::uint8Array BiometricEvaluation::Image::WSQ::getRawData () const [virtual]`

Accessor for the raw image data. The data returned should not be compressed or encoded.

Important

Bit depth of data returned from this method is at least 8. If **getBitDepth()** (p. 483) < 8, data is losslessly converted to use 8 bits to represent a single color channel.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 390)*	**Error** (p. 112) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 486).

H.162.2.2 getRawGrayscaleData()

`Memory::uint8Array BiometricEvaluation::Image::WSQ::getRawGrayscaleData (`
`uint8_t depth) const [virtual]`

Accessor for decompressed data in grayscale.

Parameters

depth	The desired bit depth of the resulting raw image. This value may either be 16, 8, or 1.

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError (p. 390)	**Error** (p. 112) decompressing image data.
Error::NotImplemented (p. 636)	Unsupported conversion based on source color depth.
Error::ParameterError (p. 655)	Invalid value for depth.

Note

This method does not save a cached copy of the decompressed image because the bit depth of the image can be changed between calls.

When depth is 1, this method returns an image that uses 8 bits to represent a single pixel. The depth parameter is used to adjust the number of gray levels. When depth is 1, there are only 2 gray levels (black and white), despite using 8 bits to represent each pixel.

Alpha channels are completely ignored when converting to grayscale.

Implements **BiometricEvaluation::Image::Image** (p. 487).

H.162.2.3 isWSQ()

```
static bool BiometricEvaluation::Image::WSQ::isWSQ (
            const uint8_t * data,
            uint64_t size)  [static]
```

Whether or not data is a **WSQ** (p. 847) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a **WSQ** (p. 847) image, false otherwise

H.163 BiometricEvaluation::Feature::Sort::XY Class Reference

`#include <be_feature_sort.h>`

Public Member Functions

- bool **operator()** (const **BiometricEvaluation::Feature::MinutiaPoint** &lhs, const **BiometricEvaluation↩ ::Feature::MinutiaPoint** &rhs) const

  MinutiaPoint (p. 619) *Cartesian X-Y ascending comparator.*

H.163.1 Detailed Description

Sort (p. 117) by increasing Cartesian X-Y coordinate

H.164 BiometricEvaluation::Feature::Sort::YX Class Reference

`#include <be_feature_sort.h>`

Public Member Functions

- bool **operator()** (const **BiometricEvaluation::Feature::MinutiaPoint** &lhs, const **BiometricEvaluation↩ ::Feature::MinutiaPoint** &rhs) const

  MinutiaPoint (p. 619) *Cartesian Y-X ascending comparator.*

H.164.1 Detailed Description

Sort (p. 117) by increasing Cartesian Y-X coordinate

Appendix I

File Documentation

I.1 be_data_interchange_an2k.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_DATA_INTERCHANGE_AN2K__
00012 #define __BE_DATA_INTERCHANGE_AN2K__
00013
00014 #include <map>
00015 #include <set>
00016 #include <string>
00017 #include <vector>
00018
00019 #include <be_finger_an2kminutiae_data_record.h>
00020 #include <be_finger_an2kview_fixedres.h>
00021 #include <be_latent_an2kview.h>
00022 #include <be_finger_an2kview_capture.h>
00023 #include <be_io_utility.h>
00024 #include <be_memory_autobuffer.h>
00025 #include <be_palm_an2kview.h>
00026
00027 namespace BiometricEvaluation
00028 {
00029     namespace DataInterchange
00030     {
00039         class AN2KRecord {
00040         public:
00046             struct DomainName {
00058                 DomainName(
00059                     std::string identifier = "",
00060                     std::string version = "") :
00061                     identifier(identifier),
00062                     version(version) {};
00063
00068                 std::string identifier;
00070                 std::string version;
00071             };
00073             using DomainName = struct DomainName;
00074
00075             struct CharacterSet {
00088                 CharacterSet(
00089                     uint16_t identifier = 0,
00090                     std::string commonName = "",
00091                     std::string version = "") :
00092                     identifier(identifier),
00093                     commonName(commonName),
00094                     version(version) {};
```

```
00095
00097                uint16_t identifier;
00099                std::string commonName;
00101                std::string version;
00102            };
00104            using CharacterSet = struct CharacterSet;
00105
00106
00125            static std::set<int>
00126            recordLocations(
00127                Memory::uint8Array &buf,
00128                const View::AN2KView::RecordType recordType);
00129
00144            static std::set<int>
00145            recordLocations(
00146                const ANSI_NIST *an2k,
00147                const View::AN2KView::RecordType recordType);
00148
00163            AN2KRecord(
00164                const std::string filename);
00165
00177            AN2KRecord(
00178                Memory::uint8Array &buf);
00179
00184            std::string getVersionNumber() const;
00185
00190            std::string getDate() const;
00191
00196            std::string getDestinationAgency() const;
00197
00202            std::string getOriginatingAgency() const;
00203
00208            std::string getTransactionControlNumber() const;
00209
00214            std::string getNativeScanningResolution() const;
00215
00220            std::string getNominalTransmittingResolution() const;
00221
00228            uint32_t getFingerLatentCount() const;
00229
00240            std::vector<Latent::AN2KView>
00241                getFingerLatents() const;
00242
00249            uint32_t getFingerCaptureCount() const;
00250
00261            std::vector<Finger::AN2KViewCapture>
00262                getFingerCaptures() const;
00263
00273            uint32_t
00274            getFingerFixedResolutionCaptureCount()
00275                const;
00276
00277
00290            uint32_t
00291            getFingerFixedResolutionCaptureCount(
00292                const View::AN2KView::RecordType type)
00293                const;
00294
00309            std::vector<Finger::AN2KViewFixedResolution>
00310            getFingerFixedResolutionCaptures()
00311                const;
00312
00313
00330            std::vector<Finger::AN2KViewFixedResolution>
00331            getFingerFixedResolutionCaptures(
00332                const View::AN2KView::RecordType type)
00333                const;
00334
00342            uint32_t
00343            getPalmCaptureCount()
00344                const;
00345
00357            std::vector<Palm::AN2KView>
00358            getPalmCaptures()
00359                const;
00360
00368            std::vector<Finger::AN2KMinutiaeDataRecord>
```

```
00369             getMinutiaeDataRecordSet()
00370                 const;
00371
00379             uint8_t
00380             getPriority()
00381                 const;
00382
00392             DomainName
00393             getDomainName()
00394                 const;
00395
00404             struct tm
00405             getGreenwichMeanTime()
00406                 const;
00407
00417             std::vector<CharacterSet>
00418             getDirectoryOfCharacterSets()
00419                 const;
00420
00433             static bool
00434             isAN2KRecord(
00435                 const std::string &filename);
00436
00449             static bool
00450             isAN2KRecord(
00451                 BiometricEvaluation::Memory::uint8Array &buf);
00452
00453         protected:
00454             AN2KRecord() { }
00455
00456         private:
00457             std::string _version;
00458             std::string _date;
00459             std::string _dai;
00460             std::string _ori;
00461             std::string _tcn;
00462             std::string _nsr;
00463             std::string _ntr;
00465             uint8_t _pry;
00467             std::string _tcr;
00469             DomainName _domainName;
00471             struct tm _gmt;
00473             std::vector<CharacterSet> _dcs;
00474
00475             std::map<View::AN2KView::RecordType,
00476                 std::vector<Finger::AN2KViewFixedResolution>>
00477                 _fingerFixedResolutionCaptures;
00478             std::vector<Latent::AN2KView> _fingerLatents;
00479             std::vector<Finger::AN2KViewCapture> _fingerCaptures;
00480             std::vector<Palm::AN2KView> _palmCaptures;
00482             std::vector<Finger::AN2KMinutiaeDataRecord>
00483                 _minutiaeDataRecordSet;
00484
00493             void readAN2KRecord(Memory::uint8Array &buf);
00494             void readType1Record(Memory::uint8Array &buf);
00495
00503                 void readMinutiaeData(Memory::uint8Array &buf);
00504             void readFingerCaptures(Memory::uint8Array &buf);
00505             void readFingerLatents(Memory::uint8Array &buf);
00506             void readFixedResolutionCaptures(Memory::uint8Array
00507                 &buf);
00508             void readPalmCaptures(Memory::uint8Array &buf);
00509         };
00510     }
00511 }
00512 #endif
00513
```

I.2 be_data_interchange_ansi2004.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
```

```
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_DATA_INTERCHANGE_ANSI2004_H_
00012 #define BE_DATA_INTERCHANGE_ANSI2004_H_
00013
00014 #include <vector>
00015
00016 #include <be_feature_incitsminutiae.h>
00017 #include <be_finger_ansi2004view.h>
00018
00019 namespace BiometricEvaluation
00020 {
00021     namespace DataInterchange
00022     {
00024         class ANSI2004Record
00025         {
00026         public:
00041             ANSI2004Record(
00042                 const BiometricEvaluation::Memory::uint8Array &fmr,
00043                 const BiometricEvaluation::Memory::uint8Array &fir);
00044
00059             ANSI2004Record(
00060                 const std::string &fmrPath,
00061                 const std::string &firPath);
00062
00071             ANSI2004Record(
00072                 const
00073                 std::initializer_list<
00074                 BiometricEvaluation::Finger::ANSI2004View> &views);
00075
00089             Finger::ANSI2004View
00090             getView(
00091                 const uint64_t viewNumber)
00092                 const;
00093
00105             uint64_t
00106             insertView(
00107                 const Finger::ANSI2004View &view);
00108
00125             uint64_t
00126             insertView(
00127                 const Finger::ANSI2004View &view,
00128                 const uint64_t viewNumber);
00129
00145             uint64_t
00146             updateView(
00147                 const Finger::ANSI2004View &view,
00148                 const uint64_t viewNumber);
00149
00163             void
00164             removeView(
00165                 const uint64_t viewNumber);
00166
00180             void
00181             isolateView(
00182                 const uint64_t viewNumber);
00183
00192             std::vector<
00193                 BiometricEvaluation::Feature::INCITSMinutiae>
00194             getMinutia()
00195                 const;
00196
00207             BiometricEvaluation::Feature::INCITSMinutiae
00208             getMinutia(
00209                 uint32_t viewNumber)
00210                 const;
00211
00223             void
00224             setMinutia(
00225                 const std::vector<
00226                 BiometricEvaluation::Feature::INCITSMinutiae>
00227                 &minutia);
00228
00241             void
```

```
00242             setMinutia(
00243                 uint32_t viewNumber,
00244                 const BiometricEvaluation::Feature::INCITSMinutiae
00245                 &minutia);
00246
00257             BiometricEvaluation::Memory::uint8Array
00258             getFMR()
00259                 const;
00260
00270             uint64_t
00271             getNumFingerViews()
00272                 const;
00273         protected:
00274
00291             uint64_t
00292             getFMRLength()
00293                 const;
00294
00310             uint64_t
00311             getEDBLength()
00312                 const;
00313
00314         private:
00316             std::vector<BiometricEvaluation::Finger::ANSI2004View>
00317                 _views;
00318
00323             void init(
00324                 const Memory::uint8Array &fmr,
00325                 const Memory::uint8Array &fir);
00326         };
00327     }
00328 }
00329
00330 #endif /* BE_DATA_INTERCHANGE_ANSI2004_H_ */
```

I.3 be_data_interchange_finger.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_DATA_INTERCHANGE_FINGER__
00012 #define BE_DATA_INTERCHANGE_FINGER__
00013
00014 #include <be_data_interchange_ansi2004.h>
00015 #include <be_feature_sort.h>
00016 #include <be_image.h>
00017
00018 namespace BiometricEvaluation
00019 {
00020     namespace DataInterchange
00021     {
00022         namespace Finger
00023         {
00059             Memory::uint8Array
00060             ANSI2004ToISOCard2011(
00061                 const DataInterchange::ANSI2004Record &ansi2004,
00062                 const uint32_t viewNumber = 1,
00063                 const uint8_t maximumMinutia = 60,
00064                 const uint8_t minimumMinutia = 0,
00065                 const Feature::Sort::Kind &sortOrder =
00066                 Feature::Sort::Kind::QualityDescending);
00067
00103             Memory::uint8Array
00104             ANSI2004ToISOCard2011(
00105                 const Memory::uint8Array &ansi2004,
00106                 const uint32_t viewNumber = 1,
00107                 const uint8_t maximumMinutia = 60,
00108                 const uint8_t minimumMinutia = 0,
```

```
00109                 const Feature::Sort::Kind &sortOrder =
00110                 Feature::Sort::Kind::QualityDescending);
00111         }
00112     }
00113 }
00114
00115 #endif /* BE_DATA_INTERCHANGE_FINGER__ */
```

I.4 be_device_smartcard.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_DEVICE_SMARTCARD_H__
00012 #define __BE_DEVICE_SMARTCARD_H__
00013
00014 #include <cstdint>
00015 #include <memory>
00016 #include <string>
00017 #include <be_memory_autoarray.h>
00018
00019 namespace BiometricEvaluation
00020 {
00021     namespace Device
00022     {
00031         class Smartcard {
00032         public:
00033             class APDU; /* Represents an APDU sent to a card */
00034
00040             struct APDUResponse {
00042                 uint8_t sw1{0};
00043
00045                 uint8_t sw2{0};
00046
00048                 Memory::uint8Array data;
00049
00051                 APDUResponse() = default;
00052
00061                 APDUResponse(
00062                     const Memory::uint8Array &data,
00063                     const uint8_t sw1,
00064                     const uint8_t sw2);
00065
00066                 ~APDUResponse() = default;
00067             };
00068
00079             struct APDUException {
00084                 APDUResponse response;
00085
00089                 Memory::uint8Array apdu;
00090
00092                 APDUException() = default;
00093
00100                 APDUException(
00101                     const APDUResponse &response,
00102                     const Memory::uint8Array &apdu);
00103             };
00104
00120             Smartcard(
00121                 unsigned int cardNum);
00122
00147             Smartcard(
00148                 unsigned int cardNum,
00149                 const Memory::uint8Array &appID);
00150
00176             Memory::uint8Array getDedicatedFileObject(
00177                 const Memory::uint8Array &objectID);
00178
```

```
00196             APDUResponse sendAPDU(
00197                 Device::Smartcard::APDU &apdu);
00198
00204             Memory::uint8Array getLastAPDU() const;
00205
00213             Memory::uint8Array getLastResponseData() const;
00214
00223             std::string getReaderID() const;
00224
00232             void setDryrun(bool state);
00233
00234             /*
00235              * We cannot use the default destructor here due to the
00236              * Impl smart pointer contained within this object.
00237              */
00241             ~Smartcard();
00242
00252              Smartcard(Smartcard&& other) noexcept;
00253
00263              Smartcard& operator=(Smartcard&& other) noexcept;
00264         private:
00265             class Impl;
00266             std::unique_ptr<Smartcard::Impl> pimpl;
00267         };
00268     }
00269 }
00270 #endif /* __BE_DEVICE_SMARTCARD_H__ */
00271
```

I.5 be_device_smartcard_apdu.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_DEVICE_SMARTCARD_APDU_H__
00012 #define __BE_DEVICE_SMARTCARD_APDU_H__
00013
00014 #include <cstdint>
00015 #include <string>
00016 #include <be_device_smartcard.h>
00017
00018 namespace BiometricEvaluation
00019 {
00020     namespace Device
00021     {
00025         class Smartcard::APDU {
00026         public:
00027             /* Bit masks to indicate optional fields to include */
00028
00030             static const int FIELD_LC{0x00000001};
00032             static const int FIELD_LE{0x00000002};
00033
00034             /* Define the length of the APDU fields */
00035             static const int FLEN_CLA{1};
00036             static const int FLEN_INS{1};
00037             static const int FLEN_P1{1};
00038             static const int FLEN_P2{1};
00039             static const int FLEN_LC_SHORT{1};
00040             static const int FLEN_LC_EXTENDED{3};
00041             static const int FLEN_LE_SHORT{1};
00042             static const int FLEN_LE_EXTENDED{3};
00043             static const int FLEN_TRAILER{2};
00044
00045             static const int FLAG_CLA_NOCHAIN{0x00};
00046             static const int FLAG_CLA_CHAIN{0x10};
00047
00048             /*
00049              * The max size of any command data is determined by
```

```
00050              * the max size of the Le field, and that is 0 (absent),
00051              * 1, or 3 bytes. In the 3-byte case, the first byte is
00052              * 0x00, and the next two are 0x0001-0xFFFF.
00053              * The same approach is used for the expected response
00054              * Le field.
00055              */
00056             static const int MAX_NC_SIZE{0xFFFF};
00057             static const int MAX_LE_SIZE{0xFFFF};
00058
00059             static const int MAX_SHORT_LC{255};
00060             static const int MAX_SHORT_LE{255};
00061             static const int HEADER_LEN
00062                 {FLEN_CLA + FLEN_INS + FLEN_P1 + FLEN_P2};
00063
00064             /*
00065              * Define some response codes for SW1.
00066              */
00067             static const int NORMAL_COMPLETE{0x90};
00068             static const int NORMAL_CHAINING{0x61};
00069             static const int WARN_NVM_UNCHANGED{0x62};
00070             static const int WARN_NVM_CHANGED{0x63};
00071             static const int EXEC_ERR_NVM_UNCHANGED{0x64};
00072             static const int EXEC_ERR_NVM_CHANGED{0x65};
00073             static const int EXEC_ERR_SECURITY{0x66};
00074             static const int CHECK_ERR_WRONG_LENGTH{0x67};
00075             static const int CHECK_ERR_CLA_FUNCTION{0x68};
00076             static const int CHECK_ERR_CMD_NOT_ALLOWED{0x69};
00077             static const int CHECK_ERR_WRONG_PARAM_QUAL{0x6A};
00078             static const int CHECK_ERR_WRONG_PARAM{0x6B};
00079             static const int CHECK_ERR_WRONG_LE{0x6C};
00080             static const int CHECK_ERR_INVALID_INS{0x6D};
00081             static const int CHECK_ERR_CLA_UNSUPPORTED{0x6E};
00082             static const int CHECK_ERR_NO_DIAGNOSIS{0x6F};
00083
00084             /*
00085              * Define some response codes for SW2.
00086              */
00087             static const int NO_INFORMATION{0x00};
00088             static const int INCORRECT_PARAMETERS{0x80};
00089             static const int FUNCTION_NOT_SUPPORTED{0x81};
00090             static const int FILE_OR_APP_NOT_FOUND{0x82};
00091
00092             /*
00093              * Mask for SW2 retry counter.
00094             */
00095             static const int RETRY_COUNTER_MASK{0x0F};
00096             static const int RETRY_COUNTER_INDICATOR{0xC0};
00097             static const int RETRY_COUNTER_INDICATOR_MASK{0xF0};
00098             static const int RETRY_COUNTER_MAX{15};
00099
00100             /*
00101              * Data that makes up the actual APDU fields.
00102              */
00103
00105             uint8_t     cla;
00107             uint8_t     ins;
00109             uint8_t     p1;
00111             uint8_t     p2;
00113             uint16_t    lc;
00115             uint8_t     nc[MAX_NC_SIZE];
00117             uint16_t    le;
00119             uint8_t     field_mask;
00120         };
00121     }
00122 }
00123 #endif /* __BE_DEVICE_SMARTCARD_APDU_H__ */
00124
```

I.6 be_device_tlv.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
```

```
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_DEVICE_TLV_H__
00012 #define __BE_DEVICE_TLV_H__
00013
00014 #include <memory>
00015 #include <string>
00016 #include <vector>
00017
00018 #include <be_memory_indexedbuffer.h>
00019
00020 namespace BiometricEvaluation
00021 {
00022     namespace Device
00023     {
00034         class TLV {
00035         public:
00046             static std::string stringFromTLV(
00047                 const TLV &tlv,
00048                 const int tabCount);
00049
00057             TLV();
00058
00066             TLV(const Memory::uint8Array &buf);
00067
00074             TLV(Memory::IndexedBuffer &ibuf);
00075
00083             TLV(const std::string &filename);
00084
00098             void setTag(const Memory::uint8Array &tag);
00099
00104             const Memory::uint8Array getTag() const;
00105
00111             uint32_t getTagNum() const;
00112
00118             uint8_t getTagClass() const;
00119
00125             bool isPrimitive() const;
00126
00136             void setPrimitive(const Memory::uint8Array &value);
00137
00145             Memory::uint8Array getPrimitive() const;
00146
00154             void addChild(const TLV &tlv);
00155
00163             std::vector<TLV> getChildren() const;
00164
00176             Memory::uint8Array getRawTLV() const;
00177
00178         private:
00179             class Impl;
00180             /*
00181              * The PIMPL is a shared pointer so the implementation
00182              * can access other TLV implementation object's state.
00183              */
00184             std::shared_ptr<TLV::Impl> pimpl;
00185         };
00186     }
00187 }
00188 #endif /* __BE_DEVICE_TLV_H__ */
00189
```

I.7 be_dirent_windows.h

```
00001 /*
00002  * Dirent interface for Microsoft Visual Studio
00003  *
00004  * Copyright (C) 1998-2019 Toni Ronkko
00005  * This file is part of dirent.  Dirent may be freely distributed
00006  * under the MIT license.  For all details and documentation, see
00007  * https://github.com/tronkko/dirent
00008  */
```

```
00009 #ifndef DIRENT_H
00010 #define DIRENT_H
00011
00012  /* Hide warnings about unreferenced local functions */
00013 #if defined(__clang__)
00014 #   pragma clang diagnostic ignored "-Wunused-function"
00015 #elif defined(_MSC_VER)
00016 #   pragma warning(disable:4505)
00017 #elif defined(__GNUC__)
00018 #   pragma GCC diagnostic ignored "-Wunused-function"
00019 #endif
00020
00021 /*
00022  * Include windows.h without Windows Sockets 1.1 to prevent conflicts with
00023  * Windows Sockets 2.0.
00024  */
00025 #ifndef WIN32_LEAN_AND_MEAN
00026 #   define WIN32_LEAN_AND_MEAN
00027 #endif
00028 #include <windows.h>
00029
00030 #include <stdio.h>
00031 #include <stdarg.h>
00032 #include <wchar.h>
00033 #include <string.h>
00034 #include <stdlib.h>
00035 #include <malloc.h>
00036 #include <sys/types.h>
00037 #include <sys/stat.h>
00038 #include <errno.h>
00039
00040  /* Indicates that d_type field is available in dirent structure */
00041 #define _DIRENT_HAVE_D_TYPE
00042
00043 /* Indicates that d_namlen field is available in dirent structure */
00044 #define _DIRENT_HAVE_D_NAMLEN
00045
00046 /* Entries missing from MSVC 6.0 */
00047 #if !defined(FILE_ATTRIBUTE_DEVICE)
00048 #   define FILE_ATTRIBUTE_DEVICE 0x40
00049 #endif
00050
00051 /* File type and permission flags for stat(), general mask */
00052 #if !defined(S_IFMT)
00053 #   define S_IFMT _S_IFMT
00054 #endif
00055
00056 /* Directory bit */
00057 #if !defined(S_IFDIR)
00058 #   define S_IFDIR _S_IFDIR
00059 #endif
00060
00061 /* Character device bit */
00062 #if !defined(S_IFCHR)
00063 #   define S_IFCHR _S_IFCHR
00064 #endif
00065
00066 /* Pipe bit */
00067 #if !defined(S_IFFIFO)
00068 #   define S_IFFIFO _S_IFFIFO
00069 #endif
00070
00071 /* Regular file bit */
00072 #if !defined(S_IFREG)
00073 #   define S_IFREG _S_IFREG
00074 #endif
00075
00076 /* Read permission */
00077 #if !defined(S_IREAD)
00078 #   define S_IREAD _S_IREAD
00079 #endif
00080
00081 /* Write permission */
00082 #if !defined(S_IWRITE)
00083 #   define S_IWRITE _S_IWRITE
00084 #endif
00085
```

```
00086 /* Execute permission */
00087 #if !defined(S_IEXEC)
00088 #   define S_IEXEC _S_IEXEC
00089 #endif
00090
00091 /* Pipe */
00092 #if !defined(S_IFIFO)
00093 #   define S_IFIFO _S_IFIFO
00094 #endif
00095
00096 /* Block device */
00097 #if !defined(S_IFBLK)
00098 #   define S_IFBLK 0
00099 #endif
00100
00101 /* Link */
00102 #if !defined(S_IFLNK)
00103 #   define S_IFLNK 0
00104 #endif
00105
00106 /* Socket */
00107 #if !defined(S_IFSOCK)
00108 #   define S_IFSOCK 0
00109 #endif
00110
00111 /* Read user permission */
00112 #if !defined(S_IRUSR)
00113 #   define S_IRUSR S_IREAD
00114 #endif
00115
00116 /* Write user permission */
00117 #if !defined(S_IWUSR)
00118 #   define S_IWUSR S_IWRITE
00119 #endif
00120
00121 /* Execute user permission */
00122 #if !defined(S_IXUSR)
00123 #   define S_IXUSR 0
00124 #endif
00125
00126 /* Read group permission */
00127 #if !defined(S_IRGRP)
00128 #   define S_IRGRP 0
00129 #endif
00130
00131 /* Write group permission */
00132 #if !defined(S_IWGRP)
00133 #   define S_IWGRP 0
00134 #endif
00135
00136 /* Execute group permission */
00137 #if !defined(S_IXGRP)
00138 #   define S_IXGRP 0
00139 #endif
00140
00141 /* Read others permission */
00142 #if !defined(S_IROTH)
00143 #   define S_IROTH 0
00144 #endif
00145
00146 /* Write others permission */
00147 #if !defined(S_IWOTH)
00148 #   define S_IWOTH 0
00149 #endif
00150
00151 /* Execute others permission */
00152 #if !defined(S_IXOTH)
00153 #   define S_IXOTH 0
00154 #endif
00155
00156 /* Maximum length of file name */
00157 #if !defined(PATH_MAX)
00158 #   define PATH_MAX MAX_PATH
00159 #endif
00160 #if !defined(FILENAME_MAX)
00161 #   define FILENAME_MAX MAX_PATH
00162 #endif
```

```
00163 #if !defined(NAME_MAX)
00164 #   define NAME_MAX FILENAME_MAX
00165 #endif
00166
00167 /* File type flags for d_type */
00168 #define DT_UNKNOWN 0
00169 #define DT_REG S_IFREG
00170 #define DT_DIR S_IFDIR
00171 #define DT_FIFO S_IFIFO
00172 #define DT_SOCK S_IFSOCK
00173 #define DT_CHR S_IFCHR
00174 #define DT_BLK S_IFBLK
00175 #define DT_LNK S_IFLNK
00176
00177 /* Macros for converting between st_mode and d_type */
00178 #define IFTODT(mode) ((mode) & S_IFMT)
00179 #define DTTOIF(type) (type)
00180
00181 /*
00182  * File type macros.  Note that block devices, sockets and links cannot be
00183  * distinguished on Windows and the macros S_ISBLK, S_ISSOCK and S_ISLNK are
00184  * only defined for compatibility.  These macros should always return false
00185  * on Windows.
00186  */
00187 #if !defined(S_ISFIFO)
00188 #   define S_ISFIFO(mode) (((mode) & S_IFMT) == S_IFIFO)
00189 #endif
00190 #if !defined(S_ISDIR)
00191 #   define S_ISDIR(mode) (((mode) & S_IFMT) == S_IFDIR)
00192 #endif
00193 #if !defined(S_ISREG)
00194 #   define S_ISREG(mode) (((mode) & S_IFMT) == S_IFREG)
00195 #endif
00196 #if !defined(S_ISLNK)
00197 #   define S_ISLNK(mode) (((mode) & S_IFMT) == S_IFLNK)
00198 #endif
00199 #if !defined(S_ISSOCK)
00200 #   define S_ISSOCK(mode) (((mode) & S_IFMT) == S_IFSOCK)
00201 #endif
00202 #if !defined(S_ISCHR)
00203 #   define S_ISCHR(mode) (((mode) & S_IFMT) == S_IFCHR)
00204 #endif
00205 #if !defined(S_ISBLK)
00206 #   define S_ISBLK(mode) (((mode) & S_IFMT) == S_IFBLK)
00207 #endif
00208
00209  /* Return the exact length of the file name without zero terminator */
00210 #define _D_EXACT_NAMLEN(p) ((p)->d_namlen)
00211
00212 /* Return the maximum size of a file name */
00213 #define _D_ALLOC_NAMLEN(p) ((PATH_MAX)+1)
00214
00215
00216 #ifdef __cplusplus
00217 extern "C" {
00218 #endif
00219
00220
00221     /* Wide-character version */
00222     struct _wdirent {
00223         /* Always zero */
00224         long d_ino;
00225
00226         /* File position within stream */
00227         long d_off;
00228
00229         /* Structure size */
00230         unsigned short d_reclen;
00231
00232         /* Length of name without \0 */
00233         size_t d_namlen;
00234
00235         /* File type */
00236         int d_type;
00237
00238         /* File name */
00239         wchar_t d_name[PATH_MAX + 1];
```

```
00240     };
00241     typedef struct _wdirent _wdirent;
00242
00243     struct _WDIR {
00244         /* Current directory entry */
00245         struct _wdirent ent;
00246
00247         /* Private file data */
00248         WIN32_FIND_DATAW data;
00249
00250         /* True if data is valid */
00251         int cached;
00252
00253         /* Win32 search handle */
00254         HANDLE handle;
00255
00256         /* Initial directory name */
00257         wchar_t* patt;
00258     };
00259     typedef struct _WDIR _WDIR;
00260
00261     /* Multi-byte character version */
00262     struct dirent {
00263         /* Always zero */
00264         long d_ino;
00265
00266         /* File position within stream */
00267         long d_off;
00268
00269         /* Structure size */
00270         unsigned short d_reclen;
00271
00272         /* Length of name without \0 */
00273         size_t d_namlen;
00274
00275         /* File type */
00276         int d_type;
00277
00278         /* File name */
00279         char d_name[PATH_MAX + 1];
00280     };
00281     typedef struct dirent dirent;
00282
00283     struct DIR {
00284         struct dirent ent;
00285         struct _WDIR* wdirp;
00286     };
00287     typedef struct DIR DIR;
00288
00289
00290     /* Dirent functions */
00291     static DIR* opendir(const char* dirname);
00292     static _WDIR* _wopendir(const wchar_t* dirname);
00293
00294     static struct dirent* readdir(DIR* dirp);
00295     static struct _wdirent* _wreaddir(_WDIR* dirp);
00296
00297     static int readdir_r(
00298         DIR* dirp, struct dirent* entry, struct dirent** result);
00299     static int _wreaddir_r(
00300         _WDIR* dirp, struct _wdirent* entry, struct _wdirent** result);
00301
00302     static int closedir(DIR* dirp);
00303     static int _wclosedir(_WDIR* dirp);
00304
00305     static void rewinddir(DIR* dirp);
00306     static void _wrewinddir(_WDIR* dirp);
00307
00308     static int scandir(const char* dirname, struct dirent*** namelist,
00309         int (*filter)(const struct dirent*),
00310         int (*compare)(const struct dirent**, const struct dirent**));
00311
00312     static int alphasort(const struct dirent** a, const struct dirent** b);
00313
00314     static int versionsort(const struct dirent** a, const struct dirent** b);
00315
00316
```

```
00317     /* For compatibility with Symbian */
00318 #define wdirent _wdirent
00319 #define WDIR _WDIR
00320 #define wopendir _wopendir
00321 #define wreaddir _wreaddir
00322 #define wclosedir _wclosedir
00323 #define wrewinddir _wrewinddir
00324
00325
00326 /* Internal utility functions */
00327     static WIN32_FIND_DATAW* dirent_first(_WDIR* dirp);
00328     static WIN32_FIND_DATAW* dirent_next(_WDIR* dirp);
00329
00330     static int dirent_mbstowcs_s(
00331         size_t* pReturnValue,
00332         wchar_t* wcstr,
00333         size_t sizeInWords,
00334         const char* mbstr,
00335         size_t count);
00336
00337     static int dirent_wcstombs_s(
00338         size_t* pReturnValue,
00339         char* mbstr,
00340         size_t sizeInBytes,
00341         const wchar_t* wcstr,
00342         size_t count);
00343
00344     static void dirent_set_errno(int error);
00345
00346
00347     /*
00348      * Open directory stream DIRNAME for read and return a pointer to the
00349      * internal working area that is used to retrieve individual directory
00350      * entries.
00351      */
00352     static _WDIR*
00353         _wopendir(
00354             const wchar_t* dirname)
00355     {
00356         _WDIR* dirp;
00357 #if WINAPI_FAMILY_PARTITION(WINAPI_PARTITION_DESKTOP)
00358         /* Desktop */
00359         DWORD n;
00360 #else
00361         /* WinRT */
00362         size_t n;
00363 #endif
00364         wchar_t* p;
00365
00366         /* Must have directory name */
00367         if (dirname == NULL || dirname[0] == '\0') {
00368             dirent_set_errno(ENOENT);
00369             return NULL;
00370         }
00371
00372         /* Allocate new _WDIR structure */
00373         dirp = (_WDIR*)malloc(sizeof(struct _WDIR));
00374         if (!dirp) {
00375             return NULL;
00376         }
00377
00378         /* Reset _WDIR structure */
00379         dirp->handle = INVALID_HANDLE_VALUE;
00380         dirp->patt = NULL;
00381         dirp->cached = 0;
00382
00383         /*
00384          * Compute the length of full path plus zero terminator
00385          *
00386          * Note that on WinRT there's no way to convert relative paths
00387          * into absolute paths, so just assume it is an absolute path.
00388          */
00389 #if WINAPI_FAMILY_PARTITION(WINAPI_PARTITION_DESKTOP)
00390          /* Desktop */
00391         n = GetFullPathNameW(dirname, 0, NULL, NULL);
00392 #else
00393          /* WinRT */
```

```
00394         n = wcslen(dirname);
00395 #endif
00396 
00397         /* Allocate room for absolute directory name and search pattern */
00398         dirp->patt = (wchar_t*)malloc(sizeof(wchar_t) * n + 16);
00399         if (dirp->patt == NULL) {
00400             goto exit_closedir;
00401         }
00402 
00403         /*
00404          * Convert relative directory name to an absolute one.  This
00405          * allows rewinddir() to function correctly even when current
00406          * working directory is changed between opendir() and rewinddir().
00407          *
00408          * Note that on WinRT there's no way to convert relative paths
00409          * into absolute paths, so just assume it is an absolute path.
00410          */
00411 #if WINAPI_FAMILY_PARTITION(WINAPI_PARTITION_DESKTOP)
00412          /* Desktop */
00413         n = GetFullPathNameW(dirname, n, dirp->patt, NULL);
00414         if (n <= 0) {
00415             goto exit_closedir;
00416         }
00417 #else
00418          /* WinRT */
00419         wcsncpy_s(dirp->patt, n + 1, dirname, n);
00420 #endif
00421 
00422         /* Append search pattern \* to the directory name */
00423         p = dirp->patt + n;
00424         switch (p[-1]) {
00425         case '\\':
00426         case '/':
00427         case ':':
00428             /* Directory ends in path separator, e.g. c:\temp\ */
00429             /*NOP*/;
00430             break;
00431 
00432         default:
00433             /* Directory name doesn't end in path separator */
00434             *p++ = '\\';
00435         }
00436         *p++ = '*';
00437         *p = '\0';
00438 
00439         /* Open directory stream and retrieve the first entry */
00440         if (!dirent_first(dirp)) {
00441             goto exit_closedir;
00442         }
00443 
00444         /* Success */
00445         return dirp;
00446 
00447         /* Failure */
00448     exit_closedir:
00449         _wclosedir(dirp);
00450         return NULL;
00451     }
00452 
00453     /*
00454      * Read next directory entry.
00455      *
00456      * Returns pointer to static directory entry which may be overwritten by
00457      * subsequent calls to _wreaddir().
00458      */
00459     static struct _wdirent*
00460         _wreaddir(
00461             _WDIR* dirp)
00462     {
00463         struct _wdirent* entry;
00464 
00465         /*
00466          * Read directory entry to buffer.  We can safely ignore the return value
00467          * as entry will be set to NULL in case of error.
00468          */
00469         (void)_wreaddir_r(dirp, &dirp->ent, &entry);
00470 
```

```
00471         /* Return pointer to statically allocated directory entry */
00472         return entry;
00473     }
00474
00475     /*
00476      * Read next directory entry.
00477      *
00478      * Returns zero on success.  If end of directory stream is reached, then sets
00479      * result to NULL and returns zero.
00480      */
00481     static int
00482         _wreaddir_r(
00483             _WDIR* dirp,
00484             struct _wdirent* entry,
00485             struct _wdirent** result)
00486     {
00487         WIN32_FIND_DATAW* datap;
00488
00489         /* Read next directory entry */
00490         datap = dirent_next(dirp);
00491         if (datap) {
00492             size_t n;
00493             DWORD attr;
00494
00495             /*
00496              * Copy file name as wide-character string.  If the file name is too
00497              * long to fit in to the destination buffer, then truncate file name
00498              * to PATH_MAX characters and zero-terminate the buffer.
00499              */
00500             n = 0;
00501             while (n < PATH_MAX && datap->cFileName[n] != 0) {
00502                 entry->d_name[n] = datap->cFileName[n];
00503                 n++;
00504             }
00505             entry->d_name[n] = 0;
00506
00507             /* Length of file name excluding zero terminator */
00508             entry->d_namlen = n;
00509
00510             /* File type */
00511             attr = datap->dwFileAttributes;
00512             if ((attr & FILE_ATTRIBUTE_DEVICE) != 0) {
00513                 entry->d_type = DT_CHR;
00514             }
00515             else if ((attr & FILE_ATTRIBUTE_DIRECTORY) != 0) {
00516                 entry->d_type = DT_DIR;
00517             }
00518             else {
00519                 entry->d_type = DT_REG;
00520             }
00521
00522             /* Reset dummy fields */
00523             entry->d_ino = 0;
00524             entry->d_off = 0;
00525             entry->d_reclen = sizeof(struct _wdirent);
00526
00527             /* Set result address */
00528             *result = entry;
00529
00530         }
00531         else {
00532
00533             /* Return NULL to indicate end of directory */
00534             *result = NULL;
00535
00536         }
00537
00538         return /*OK*/0;
00539     }
00540
00541     /*
00542      * Close directory stream opened by opendir() function.  This invalidates the
00543      * DIR structure as well as any directory entry read previously by
00544      * _wreaddir().
00545      */
00546     static int
00547         _wclosedir(
```

```
00548           _WDIR* dirp)
00549   {
00550       int ok;
00551       if (dirp) {
00552
00553           /* Release search handle */
00554           if (dirp->handle != INVALID_HANDLE_VALUE) {
00555               FindClose(dirp->handle);
00556           }
00557
00558           /* Release search pattern */
00559           free(dirp->patt);
00560
00561           /* Release directory structure */
00562           free(dirp);
00563           ok = /*success*/0;
00564
00565       }
00566       else {
00567
00568           /* Invalid directory stream */
00569           dirent_set_errno(EBADF);
00570           ok = /*failure*/-1;
00571
00572       }
00573       return ok;
00574   }
00575
00576   /*
00577    * Rewind directory stream such that _wreaddir() returns the very first
00578    * file name again.
00579    */
00580   static void
00581       _wrewinddir(
00582           _WDIR* dirp)
00583   {
00584       if (dirp) {
00585           /* Release existing search handle */
00586           if (dirp->handle != INVALID_HANDLE_VALUE) {
00587               FindClose(dirp->handle);
00588           }
00589
00590           /* Open new search handle */
00591           dirent_first(dirp);
00592       }
00593   }
00594
00595   /* Get first directory entry (internal) */
00596   static WIN32_FIND_DATAW*
00597       dirent_first(
00598           _WDIR* dirp)
00599   {
00600       WIN32_FIND_DATAW* datap;
00601       DWORD error;
00602
00603       /* Open directory and retrieve the first entry */
00604       dirp->handle = FindFirstFileExW(
00605           dirp->patt, FindExInfoStandard, &dirp->data,
00606           FindExSearchNameMatch, NULL, 0);
00607       if (dirp->handle != INVALID_HANDLE_VALUE) {
00608
00609           /* a directory entry is now waiting in memory */
00610           datap = &dirp->data;
00611           dirp->cached = 1;
00612
00613       }
00614       else {
00615
00616           /* Failed to open directory: no directory entry in memory */
00617           dirp->cached = 0;
00618           datap = NULL;
00619
00620           /* Set error code */
00621           error = GetLastError();
00622           switch (error) {
00623           case ERROR_ACCESS_DENIED:
00624               /* No read access to directory */
```

```
00625                 dirent_set_errno(EACCES);
00626                 break;
00627
00628             case ERROR_DIRECTORY:
00629                 /* Directory name is invalid */
00630                 dirent_set_errno(ENOTDIR);
00631                 break;
00632
00633             case ERROR_PATH_NOT_FOUND:
00634             default:
00635                 /* Cannot find the file */
00636                 dirent_set_errno(ENOENT);
00637             }
00638
00639         }
00640         return datap;
00641     }
00642
00643     /*
00644      * Get next directory entry (internal).
00645      *
00646      * Returns
00647      */
00648     static WIN32_FIND_DATAW*
00649         dirent_next(
00650             _WDIR* dirp)
00651     {
00652         WIN32_FIND_DATAW* p;
00653
00654         /* Get next directory entry */
00655         if (dirp->cached != 0) {
00656
00657             /* A valid directory entry already in memory */
00658             p = &dirp->data;
00659             dirp->cached = 0;
00660
00661         }
00662         else if (dirp->handle != INVALID_HANDLE_VALUE) {
00663
00664             /* Get the next directory entry from stream */
00665             if (FindNextFileW(dirp->handle, &dirp->data) != FALSE) {
00666                 /* Got a file */
00667                 p = &dirp->data;
00668             }
00669             else {
00670                 /* The very last entry has been processed or an error occurred */
00671                 FindClose(dirp->handle);
00672                 dirp->handle = INVALID_HANDLE_VALUE;
00673                 p = NULL;
00674             }
00675
00676         }
00677         else {
00678
00679             /* End of directory stream reached */
00680             p = NULL;
00681
00682         }
00683
00684         return p;
00685     }
00686
00687     /*
00688      * Open directory stream using plain old C-string.
00689      */
00690     static DIR*
00691         opendir(
00692             const char* dirname)
00693     {
00694         struct DIR* dirp;
00695
00696         /* Must have directory name */
00697         if (dirname == NULL || dirname[0] == '\0') {
00698             dirent_set_errno(ENOENT);
00699             return NULL;
00700         }
00701
```

```
00702         /* Allocate memory for DIR structure */
00703         dirp = (DIR*)malloc(sizeof(struct DIR));
00704         if (!dirp) {
00705             return NULL;
00706         }
00707         {
00708             int error;
00709             wchar_t wname[PATH_MAX + 1];
00710             size_t n;
00711 
00712             /* Convert directory name to wide-character string */
00713             error = dirent_mbstowcs_s(
00714                 &n, wname, PATH_MAX + 1, dirname, PATH_MAX + 1);
00715             if (error) {
00716                 /*
00717                  * Cannot convert file name to wide-character string.  This
00718                  * occurs if the string contains invalid multi-byte sequences or
00719                  * the output buffer is too small to contain the resulting
00720                  * string.
00721                  */
00722                 goto exit_free;
00723             }
00724 
00725 
00726             /* Open directory stream using wide-character name */
00727             dirp->wdirp = _wopendir(wname);
00728             if (!dirp->wdirp) {
00729                 goto exit_free;
00730             }
00731 
00732         }
00733 
00734         /* Success */
00735         return dirp;
00736 
00737         /* Failure */
00738     exit_free:
00739         free(dirp);
00740         return NULL;
00741     }
00742 
00743     /*
00744      * Read next directory entry.
00745      */
00746     static struct dirent*
00747         readdir(
00748             DIR* dirp)
00749     {
00750         struct dirent* entry;
00751 
00752         /*
00753          * Read directory entry to buffer.  We can safely ignore the return value
00754          * as entry will be set to NULL in case of error.
00755          */
00756         (void)readdir_r(dirp, &dirp->ent, &entry);
00757 
00758         /* Return pointer to statically allocated directory entry */
00759         return entry;
00760     }
00761 
00762     /*
00763      * Read next directory entry into called-allocated buffer.
00764      *
00765      * Returns zero on success.  If the end of directory stream is reached, then
00766      * sets result to NULL and returns zero.
00767      */
00768     static int
00769         readdir_r(
00770             DIR* dirp,
00771             struct dirent* entry,
00772             struct dirent** result)
00773     {
00774         WIN32_FIND_DATAW* datap;
00775 
00776         /* Read next directory entry */
00777         datap = dirent_next(dirp->wdirp);
00778         if (datap) {
```

```
00779             size_t n;
00780             int error;
00781 
00782             /* Attempt to convert file name to multi-byte string */
00783             error = dirent_wcstombs_s(
00784                 &n, entry->d_name, PATH_MAX + 1, datap->cFileName, PATH_MAX + 1);
00785 
00786             /*
00787              * If the file name cannot be represented by a multi-byte string,
00788              * then attempt to use old 8+3 file name.  This allows traditional
00789              * Unix-code to access some file names despite of unicode
00790              * characters, although file names may seem unfamiliar to the user.
00791              *
00792              * Be ware that the code below cannot come up with a short file
00793              * name unless the file system provides one.  At least
00794              * VirtualBox shared folders fail to do this.
00795              */
00796             if (error && datap->cAlternateFileName[0] != '\0') {
00797                 error = dirent_wcstombs_s(
00798                     &n, entry->d_name, PATH_MAX + 1,
00799                     datap->cAlternateFileName, PATH_MAX + 1);
00800             }
00801 
00802             if (!error) {
00803                 DWORD attr;
00804 
00805                 /* Length of file name excluding zero terminator */
00806                 entry->d_namlen = n - 1;
00807 
00808                 /* File attributes */
00809                 attr = datap->dwFileAttributes;
00810                 if ((attr & FILE_ATTRIBUTE_DEVICE) != 0) {
00811                     entry->d_type = DT_CHR;
00812                 }
00813                 else if ((attr & FILE_ATTRIBUTE_DIRECTORY) != 0) {
00814                     entry->d_type = DT_DIR;
00815                 }
00816                 else {
00817                     entry->d_type = DT_REG;
00818                 }
00819 
00820                 /* Reset dummy fields */
00821                 entry->d_ino = 0;
00822                 entry->d_off = 0;
00823                 entry->d_reclen = sizeof(struct dirent);
00824 
00825             }
00826             else {
00827 
00828                 /*
00829                  * Cannot convert file name to multi-byte string so construct
00830                  * an erroneous directory entry and return that.  Note that
00831                  * we cannot return NULL as that would stop the processing
00832                  * of directory entries completely.
00833                  */
00834                 entry->d_name[0] = '?';
00835                 entry->d_name[1] = '\0';
00836                 entry->d_namlen = 1;
00837                 entry->d_type = DT_UNKNOWN;
00838                 entry->d_ino = 0;
00839                 entry->d_off = -1;
00840                 entry->d_reclen = 0;
00841 
00842             }
00843 
00844             /* Return pointer to directory entry */
00845             *result = entry;
00846 
00847         }
00848         else {
00849 
00850             /* No more directory entries */
00851             *result = NULL;
00852 
00853         }
00854 
00855         return /*OK*/0;
```

```
00856     }
00857
00858     /*
00859      * Close directory stream.
00860      */
00861     static int
00862         closedir(
00863             DIR* dirp)
00864     {
00865         int ok;
00866         if (dirp) {
00867
00868             /* Close wide-character directory stream */
00869             ok = _wclosedir(dirp->wdirp);
00870             dirp->wdirp = NULL;
00871
00872             /* Release multi-byte character version */
00873             free(dirp);
00874
00875         }
00876         else {
00877
00878             /* Invalid directory stream */
00879             dirent_set_errno(EBADF);
00880             ok = /*failure*/-1;
00881
00882         }
00883         return ok;
00884     }
00885
00886     /*
00887      * Rewind directory stream to beginning.
00888      */
00889     static void
00890         rewinddir(
00891             DIR* dirp)
00892     {
00893         /* Rewind wide-character string directory stream */
00894         _wrewinddir(dirp->wdirp);
00895     }
00896
00897     /*
00898      * Scan directory for entries.
00899      */
00900     static int
00901         scandir(
00902             const char* dirname,
00903             struct dirent*** namelist,
00904             int (*filter)(const struct dirent*),
00905             int (*compare)(const struct dirent**, const struct dirent**))
00906     {
00907         struct dirent** files = NULL;
00908         size_t size = 0;
00909         size_t allocated = 0;
00910         const size_t init_size = 1;
00911         DIR* dir = NULL;
00912         struct dirent* entry;
00913         struct dirent* tmp = NULL;
00914         size_t i;
00915         int result = 0;
00916
00917         /* Open directory stream */
00918         dir = opendir(dirname);
00919         if (dir) {
00920
00921             /* Read directory entries to memory */
00922             while (1) {
00923
00924                 /* Enlarge pointer table to make room for another pointer */
00925                 if (size >= allocated) {
00926                     void* p;
00927                     size_t num_entries;
00928
00929                     /* Compute number of entries in the enlarged pointer table */
00930                     if (size < init_size) {
00931                         /* Allocate initial pointer table */
00932                         num_entries = init_size;
```

```
00933                     }
00934                     else {
00935                         /* Double the size */
00936                         num_entries = size * 2;
00937                     }
00938 
00939                     /* Allocate first pointer table or enlarge existing table */
00940                     p = realloc(files, sizeof(void*) * num_entries);
00941                     if (p != NULL) {
00942                         /* Got the memory */
00943                         files = (dirent**)p;
00944                         allocated = num_entries;
00945                     }
00946                     else {
00947                         /* Out of memory */
00948                         result = -1;
00949                         break;
00950                     }
00951 
00952                 }
00953 
00954                 /* Allocate room for temporary directory entry */
00955                 if (tmp == NULL) {
00956                     tmp = (struct dirent*)malloc(sizeof(struct dirent));
00957                     if (tmp == NULL) {
00958                         /* Cannot allocate temporary directory entry */
00959                         result = -1;
00960                         break;
00961                     }
00962                 }
00963 
00964                 /* Read directory entry to temporary area */
00965                 if (readdir_r(dir, tmp, &entry) == /*OK*/0) {
00966 
00967                     /* Did we get an entry? */
00968                     if (entry != NULL) {
00969                         int pass;
00970 
00971                         /* Determine whether to include the entry in result */
00972                         if (filter) {
00973                             /* Let the filter function decide */
00974                             pass = filter(tmp);
00975                         }
00976                         else {
00977                             /* No filter function, include everything */
00978                             pass = 1;
00979                         }
00980 
00981                         if (pass) {
00982                             /* Store the temporary entry to pointer table */
00983                             files[size++] = tmp;
00984                             tmp = NULL;
00985 
00986                             /* Keep up with the number of files */
00987                             result++;
00988                         }
00989 
00990                     }
00991                     else {
00992 
00993                         /*
00994                          * End of directory stream reached => sort entries and
00995                          * exit.
00996                          */
00997                         qsort(files, size, sizeof(void*),
00998                             (int (*) (const void*, const void*)) compare);
00999                         break;
01000 
01001                     }
01002 
01003                 }
01004                 else {
01005                     /* Error reading directory entry */
01006                     result = /*Error*/ -1;
01007                     break;
01008                 }
01009 
```

```
01010             }
01011
01012         }
01013         else {
01014             /* Cannot open directory */
01015             result = /*Error*/ -1;
01016         }
01017
01018         /* Release temporary directory entry */
01019         free(tmp);
01020
01021         /* Release allocated memory on error */
01022         if (result < 0) {
01023             for (i = 0; i < size; i++) {
01024                 free(files[i]);
01025             }
01026             free(files);
01027             files = NULL;
01028         }
01029
01030         /* Close directory stream */
01031         if (dir) {
01032             closedir(dir);
01033         }
01034
01035         /* Pass pointer table to caller */
01036         if (namelist) {
01037             *namelist = files;
01038         }
01039         return result;
01040     }
01041
01042     /* Alphabetical sorting */
01043     static int
01044         alphasort(
01045             const struct dirent** a, const struct dirent** b)
01046     {
01047         return strcoll((*a)->d_name, (*b)->d_name);
01048     }
01049
01050     /* Sort versions */
01051     static int
01052         versionsort(
01053             const struct dirent** a, const struct dirent** b)
01054     {
01055         /* FIXME: implement strverscmp and use that */
01056         return alphasort(a, b);
01057     }
01058
01059     /* Convert multi-byte string to wide character string */
01060     static int
01061         dirent_mbstowcs_s(
01062             size_t* pReturnValue,
01063             wchar_t* wcstr,
01064             size_t sizeInWords,
01065             const char* mbstr,
01066             size_t count)
01067     {
01068         int error;
01069
01070 #if defined(_MSC_VER)  &&  _MSC_VER >= 1400
01071
01072         /* Microsoft Visual Studio 2005 or later */
01073         error = mbstowcs_s(pReturnValue, wcstr, sizeInWords, mbstr, count);
01074
01075 #else
01076
01077         /* Older Visual Studio or non-Microsoft compiler */
01078         size_t n;
01079
01080         /* Convert to wide-character string (or count characters) */
01081         n = mbstowcs(wcstr, mbstr, sizeInWords);
01082         if (!wcstr || n < count) {
01083
01084             /* Zero-terminate output buffer */
01085             if (wcstr && sizeInWords) {
01086                 if (n >= sizeInWords) {
```

```
01087                     n = sizeInWords - 1;
01088                 }
01089                 wcstr[n] = 0;
01090             }
01091 
01092             /* Length of resulting multi-byte string WITH zero terminator */
01093             if (pReturnValue) {
01094                 *pReturnValue = n + 1;
01095             }
01096 
01097             /* Success */
01098             error = 0;
01099 
01100         }
01101         else {
01102 
01103             /* Could not convert string */
01104             error = 1;
01105 
01106         }
01107 
01108 #endif
01109         return error;
01110     }
01111 
01112     /* Convert wide-character string to multi-byte string */
01113     static int
01114         dirent_wcstombs_s(
01115             size_t* pReturnValue,
01116             char* mbstr,
01117             size_t sizeInBytes, /* max size of mbstr */
01118             const wchar_t* wcstr,
01119             size_t count)
01120     {
01121         int error;
01122 
01123 #if defined(_MSC_VER)  &&  _MSC_VER >= 1400
01124 
01125         /* Microsoft Visual Studio 2005 or later */
01126         error = wcstombs_s(pReturnValue, mbstr, sizeInBytes, wcstr, count);
01127 
01128 #else
01129 
01130         /* Older Visual Studio or non-Microsoft compiler */
01131         size_t n;
01132 
01133         /* Convert to multi-byte string (or count the number of bytes needed) */
01134         n = wcstombs(mbstr, wcstr, sizeInBytes);
01135         if (!mbstr || n < count) {
01136 
01137             /* Zero-terminate output buffer */
01138             if (mbstr && sizeInBytes) {
01139                 if (n >= sizeInBytes) {
01140                     n = sizeInBytes - 1;
01141                 }
01142                 mbstr[n] = '\0';
01143             }
01144 
01145             /* Length of resulting multi-bytes string WITH zero-terminator */
01146             if (pReturnValue) {
01147                 *pReturnValue = n + 1;
01148             }
01149 
01150             /* Success */
01151             error = 0;
01152 
01153         }
01154         else {
01155 
01156             /* Cannot convert string */
01157             error = 1;
01158 
01159         }
01160 
01161 #endif
01162         return error;
01163     }
```

```
01164
01165     /* Set errno variable */
01166     static void
01167         dirent_set_errno(
01168             int error)
01169     {
01170 #if defined(_MSC_VER)  &&  _MSC_VER >= 1400
01171
01172         /* Microsoft Visual Studio 2005 and later */
01173         _set_errno(error);
01174
01175 #else
01176
01177         /* Non-Microsoft compiler or older Microsoft compiler */
01178         errno = error;
01179
01180 #endif
01181     }
01182
01183
01184 #ifdef __cplusplus
01185 }
01186 #endif
01187 #endif /*DIRENT_H*/
```

I.8 be_error.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_ERROR_UTILITY__
00012 #define __BE_ERROR_UTILITY__
00013
00014 #include <string>
00015
00016 namespace BiometricEvaluation
00017 {
00025     namespace Error
00026     {
00039         std::string
00040         errorStr(
00041             bool includeErrno = false);
00042     }
00043 }
00044 #endif
00045
```

I.9 be_error_exception.h

```
00001 /*****************************************************************************
00002 * This software was developed at the National Institute of Standards and
00003 * Technology (NIST) by employees of the Federal Government in the course
00004 * of their official duties. Pursuant to title 17 Section 105 of the
00005 * United States Code, this software is not subject to copyright protection
00006 * and is in the public domain. NIST assumes no responsibility whatsoever for
00007 * its use by other parties, and makes no guarantees, expressed or implied,
00008 * about its quality, reliability, or any other characteristic.
00009 *****************************************************************************/
00010 #ifndef __BE_EXCEPTION_H__
00011 #define __BE_EXCEPTION_H__
00012
00013 #include <string>
00014
00015 /*
00016  * Define the exception classes that will be used throughout the framework.
```

```
00017  */
00018 namespace BiometricEvaluation {
00019
00020     namespace Error {
00021
00031         class Exception : public std::exception {
00032             public:
00038                 Exception();
00039
00048                 /*
00049                  * Pass info by value so we can use move
00050                  * semantics when setting object state.
00051                  */
00052                 Exception(std::string info);
00053
00054                 virtual ~Exception() = default;
00055
00063                 const char *
00064                 what() const noexcept override;
00065
00073                 const std::string
00074                 whatString() const noexcept;
00075
00076             private:
00077                 std::string _info;
00078         };
00079
00084         class FileError : public Exception {
00085             public:
00090                 FileError();
00091
00097                 FileError(const std::string &info);
00098         };
00099
00104         class ParameterError : public Exception {
00105             public:
00110                 ParameterError();
00111
00117                 ParameterError(const std::string &info);
00118         };
00119
00125         class ConversionError : public Exception {
00126             public:
00131                 ConversionError();
00132
00138                 ConversionError(const std::string &info);
00139         };
00140
00149         class DataError : public Exception {
00150             public:
00155                 DataError();
00156
00162                 DataError(const std::string &info);
00163         };
00164
00169         class MemoryError : public Exception {
00170             public:
00175                 MemoryError();
00176
00182                 MemoryError(const std::string &info);
00183         };
00184
00189         class ObjectExists : public Exception {
00190             public:
00195                 ObjectExists();
00196
00202                 ObjectExists(const std::string &info);
00203         };
00204
00209         class ObjectDoesNotExist : public Exception {
00210             public:
00215                 ObjectDoesNotExist();
00216
00222                 ObjectDoesNotExist(const std::string &info);
00223         };
00224
00229         class ObjectIsOpen : public Exception {
```

```
00230             public:
00235                 ObjectIsOpen();
00236
00242                 ObjectIsOpen(const std::string &info);
00243         };
00244
00249         class ObjectIsClosed : public Exception {
00250             public:
00255                 ObjectIsClosed();
00256
00262                 ObjectIsClosed(const std::string &info);
00263         };
00264
00270         class StrategyError : public Exception {
00271             public:
00276                 StrategyError();
00277
00283                 StrategyError(const std::string &info);
00284         };
00285
00292         class NotImplemented : public Exception {
00293             public:
00298                 NotImplemented();
00299
00305                 NotImplemented(const std::string &info);
00306         };
00307     }
00308 }
00309 #endif  /* __BE_EXCEPTION_H__ */
```

I.10 be_error_signal_manager.h

```
00001 /*****************************************************************************
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  *****************************************************************************/
00010 #ifndef __BE_ERROR_SIGNAL_MANAGER_H__
00011 #define __BE_ERROR_SIGNAL_MANAGER_H__
00012
00013 #include <csetjmp>
00014 #include <csignal>
00015
00016 #include <be_error_exception.h>
00017
00018 /*
00019  * Macros that are used by applications to indicate the start and end of
00020  * a signal handling block.
00021  */
00022 #define BEGIN_SIGNAL_BLOCK(_sigmgr, _blockname) do {            \
00023     if (!(_sigmgr)->isEnabled())                    \
00024         break;                          \
00025     (_sigmgr)->clearSigHandled();                   \
00026     (_sigmgr)->stop();                      \
00027     if (sigsetjmp(                          \
00028         BiometricEvaluation::Error::SignalManager::_sigJumpBuf, 1) != 0) \
00029      {                                  \
00030         (_sigmgr)->setSigHandled();             \
00031         goto _blockname ## _end;                \
00032     }                                   \
00033     (_sigmgr)->start();                     \
00034 } while (0)
00035
00036 #define END_SIGNAL_BLOCK(_sigmgr, _blockname) do {           \
00037     if (!(_sigmgr)->isEnabled())                    \
00038         break;                          \
00039     _blockname ## _end:                     \
00040     (_sigmgr)->stop();                      \
00041 } while (0);
00042
00043 #define ABORT_SIGNAL_MANAGER(_sigmgr) do {              \
```

```
00044     if (!(_sigmgr)->isEnabled())                    \
00045         break;                              \
00046     (_sigmgr)->stop();                          \
00047 } while (0);
00048
00049 namespace BiometricEvaluation {
00050
00051     namespace Error {
00052
00096         class SignalManager {
00097
00098         public:
00099
00108             SignalManager();
00109
00123             SignalManager(
00124                 const sigset_t signalSet);
00125
00138             void setSignalSet(
00139                 const sigset_t signalSet);
00140
00144             void clearSignalSet();
00145
00150             void setDefaultSignalSet();
00151
00157             bool sigHandled();
00158
00170             void start();
00171
00178             void stop();
00179
00183             void setSigHandled();
00184
00188             void clearSigHandled();
00189
00200             void setEnabled(
00201                 const bool enabled);
00202
00204             bool isEnabled() const;
00205
00210             static bool _canSigJump;
00215             static sigjmp_buf _sigJumpBuf;
00216
00217         protected:
00218
00219         private:
00221             bool _enabled{true};
00222
00226             sigset_t _signalSet;
00227
00231             bool _sigHandled{false};
00232         };
00233
00234         /*
00235         * Declaration of the signal handler, a function with C linkage
00236         * that will handle all signals managed by this object,
00237         * conditionally jumping to a jump block within the application
00238         * process. This function is of no interest to applications,
00239         * which should use the BEGIN_SIGNAL_BLOCK()/END_SIGNAL_BLOCK()
00240         * macro pair to take advantage of signal handling.
00241         */
00242         extern "C" {
00243             void SignalManagerSighandler(int signo,
00244                 siginfo_t *info, void *uap);
00245         }
00246     }
00247 }
00248 #endif  /* __BE_ERROR_SIGNAL_MANAGER_H__ */
```

I.11 be_face.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
```

```
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FACE_H__
00012 #define __BE_FACE_H__
00013
00014 #include <cstdint>
00015
00016 #include <be_framework_enumeration.h>
00017
00018 namespace BiometricEvaluation
00019 {
00031     namespace Face
00032     {
00037         typedef struct {
00038             uint8_t     yaw;
00039             uint8_t     pitch;
00040             uint8_t     roll;
00041             uint8_t     yawUncertainty;
00042             uint8_t     pitchUncertainty;
00043             uint8_t     rollUncertainty;
00044         } PoseAngle;
00045
00050         enum class Gender {
00051             Unspecified = 0x00,
00052             Male = 0x01,
00053             Female = 0x02,
00054             Unknown = 0xFF
00055         };
00056
00061         enum class EyeColor {
00062             Unspecified = 0x00,
00063             Black = 0x01,
00064             Blue = 0x02,
00065             Brown = 0x03,
00066             Gray = 0x04,
00067             Green = 0x05,
00068             MultiColored = 0x06,
00069             Pink = 0x07,
00070             Unknown = 0xFF
00071         };
00072
00077         enum class HairColor {
00078             Unspecified = 0x00,
00079             Bald = 0x01,
00080             Black = 0x02,
00081             Blonde = 0x03,
00082             Brown = 0x04,
00083             Gray = 0x05,
00084             White = 0x06,
00085             Red = 0x07,
00086             Unknown = 0xFF
00087         };
00088
00093         enum class Property {
00094             Glasses = 1,
00095             Moustache = 2,
00096             Beard = 3,
00097             Teeth = 4,
00098             Blink = 5,
00099             MouthOpen = 6,
00100             LeftEyePatch = 7,
00101             RightEyePatch = 8,
00102             DarkGlasses = 9,
00103             MedicalCondition = 10
00104         };
00105
00110         enum class Expression {
00111             Unspecified = 0x0000,
00112             Neutral = 0x0001,
00113             SmileClosedJaw = 0x0002,
00114             SmileOpenJaw = 0x0003,
00115             RaisedEyebrows = 0x0004,
00116             EyesLookingAway = 0x0005,
```

```
00117             Squinting = 0x0006,
00118             Frowning = 0x0007
00119         };
00120
00125         enum class ImageType {
00126             Basic = 0x00,
00127             FullFrontal = 0x01,
00128             TokenFrontal = 0x02
00129         };
00130
00135         enum class ImageDataType {
00136             JPEG = 0x00,
00137             JPEG2000 = 0x01
00138         };
00139
00144         enum class ColorSpace {
00145             Unspecified = 0x00,
00146             RGB24 = 0x01,
00147             YUV422 = 0x02,
00148             Grayscale8 = 0x03,
00149             Other = 0x04
00150         };
00151
00156         enum class SourceType {
00157             Unspecified = 0x00,
00158             StaticPhotoUnknown = 0x01,
00159             StaticPhotoDigitalStill = 0x02,
00160             StaticPhotoScan = 0x03,
00161             VideoFrameUnknown = 0x04,
00162             VideoFrameAnalog = 0x05,
00163             VideoFrameDigital = 0x06,
00164             Unknown = 0x07
00165         };
00166     }
00167 }
00168
00169 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00170     BiometricEvaluation::Face::Gender,
00171     BE_Face_Gender_EnumToStringMap);
00172
00173 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00174     BiometricEvaluation::Face::EyeColor,
00175     BE_Face_EyeColor_EnumToStringMap);
00176
00177 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00178     BiometricEvaluation::Face::HairColor,
00179     BE_Face_HairColor_EnumToStringMap);
00180
00181 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00182     BiometricEvaluation::Face::Property,
00183     BE_Face_Property_EnumToStringMap);
00184
00185 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00186     BiometricEvaluation::Face::Expression,
00187     BE_Face_Expression_EnumToStringMap);
00188
00189 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00190     BiometricEvaluation::Face::ImageType,
00191     BE_Face_ImageType_EnumToStringMap);
00192
00193 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00194     BiometricEvaluation::Face::ImageDataType,
00195     BE_Face_ImageDataType_EnumToStringMap);
00196
00197 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00198     BiometricEvaluation::Face::ColorSpace,
00199     BE_Face_ColorSpace_EnumToStringMap);
00200
00201 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00202     BiometricEvaluation::Face::SourceType,
00203     BE_Face_SourceType_EnumToStringMap);
00204
00205 #endif /* __BE_FACE_H__ */
00206
```

I.12 be_face_incitsview.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FACE_INCITSVIEW_H__
00012 #define __BE_FACE_INCITSVIEW_H__
00013
00014 #include <vector>
00015
00016 #include <be_image.h>
00017 #include <be_face.h>
00018 #include <be_feature_mpegfacepoint.h>
00019 #include <be_memory_indexedbuffer.h>
00020 #include <be_view_view.h>
00021
00022 namespace BiometricEvaluation
00023 {
00024     namespace Face
00025     {
00029         typedef std::vector<BiometricEvaluation::Face::Property>
00030             PropertySet;
00031
00044         class INCITSView : public View::View {
00045         public:
00052             Face::Gender getGender() const;
00053
00060             Face::EyeColor getEyeColor() const;
00061
00068             Face::HairColor getHairColor() const;
00069
00076             bool propertiesConsidered() const;
00077
00084             void
00085             getPropertySet(Face::PropertySet &propertySet) const;
00086
00087             BiometricEvaluation::Face::Expression
00088             getExpression() const;
00089
00096             void getFeaturePointSet(
00097                 BiometricEvaluation::Feature::MPEGFacePointSet
00098                  &featurePointSet) const;
00099
00106             Face::ImageType getImageType() const;
00107
00114             Face::ImageDataType getImageDataType() const;
00115
00122             Face::PoseAngle getPoseAngle() const;
00123
00130             Face::ColorSpace getColorSpace() const;
00131
00138             Face::SourceType getSourceType() const;
00139
00146             uint16_t getDeviceType() const;
00147
00148         protected:
00149
00150             static const uint32_t ISO2005_STANDARD = 1;
00151             static const uint32_t BASE_FORMAT_ID = 0x46414300;
00152             /* 'F''A''C' 'nul' */
00153
00154             INCITSView();
00155
00175             INCITSView(
00176                 const std::string &filename,
00177                 const uint32_t viewNumber);
00178
00196             INCITSView(
00197                 const Memory::uint8Array &buffer,
00198                 const uint32_t viewNumber);
```

```
00199
00207             Memory::uint8Array const& getFIDData() const;
00208
00228             virtual void readHeader(
00229                 BiometricEvaluation::Memory::IndexedBuffer &buf,
00230                 const uint32_t formatStandard);
00231
00246             virtual void readFaceView(
00247                 Memory::IndexedBuffer &buf);
00248
00249         private:
00250             BiometricEvaluation::Memory::uint8Array _fid;
00251
00252             BiometricEvaluation::Feature::MPEGFacePointSet
00253                 _featurePointSet;
00254             BiometricEvaluation::Face::ImageType
00255                 _imageType;
00256             BiometricEvaluation::Face::ImageDataType
00257                 _imageDataType;
00258             BiometricEvaluation::Face::Gender _gender;
00259             BiometricEvaluation::Face::EyeColor _eyeColor;
00260             BiometricEvaluation::Face::HairColor _hairColor;
00261
00262             bool _propertiesConsidered;
00263             BiometricEvaluation::Face::PropertySet _propertySet;
00264             BiometricEvaluation::Face::Expression _expression;
00265             BiometricEvaluation::Face::PoseAngle _poseAngle;
00266
00267             BiometricEvaluation::Face::ColorSpace _colorSpace;
00268             BiometricEvaluation::Face::SourceType _sourceType;
00269
00270             uint16_t _quality;
00271             uint16_t _deviceType;
00272         };
00273     }
00274 }
00275 #endif /* __BE_FACE_INCITSVIEW_H__ */
00276
```

I.13 be_face_iso2005view.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FACE_ISO2005VIEW_H__
00012 #define __BE_FACE_ISO2005VIEW_H__
00013
00014 #include <vector>
00015
00016 #include <be_image.h>
00017 #include <be_face_incitsview.h>
00018 #include <be_memory_indexedbuffer.h>
00019 #include <be_view_view.h>
00020
00021 namespace BiometricEvaluation
00022 {
00023     namespace Face
00024     {
00033         class ISO2005View : public Face::INCITSView {
00034         public:
00039             ISO2005View();
00040
00060             ISO2005View(
00061                 const std::string &filename,
00062                 const uint32_t viewNumber);
00063
00082             ISO2005View(
00083                 const Memory::uint8Array &buffer,
```

```
00084                 const uint32_t viewNumber);
00085
00086         protected:
00087
00088             static const uint32_t BASE_SPEC_VERSION = 0x30313000;
00089             /* '0''1''0' 'nul' */
00090
00102             void readISOHeader(
00103                 BiometricEvaluation::Memory::IndexedBuffer &buf);
00104
00105         private:
00106         };
00107     }
00108 }
00109 #endif /* __BE_FACE_ISO2005VIEW_H__ */
00110
```

I.14 be_feature.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FEATURE_H__
00012 #define __BE_FEATURE_H__
00013
00014 #include <vector>
00015 #include <be_finger.h>
00016 #include <be_palm.h>
00017 #include <be_plantar.h>
00018
00019 namespace BiometricEvaluation
00020 {
00026     namespace Feature
00027     {
00033         enum class PositionType {
00034             Finger      = 0,
00035             Palm        = 1,
00036             Plantar     = 2
00037         };
00038
00048         struct FrictionRidgeGeneralizedPosition {
00049             PositionType        posType;
00050             union {
00051                 Finger::Position    fingerPos;
00052                 Palm::Position        palmPos;
00053                 Plantar::Position  plantarPos;
00054             } position;
00055         };
00056         using FGP = struct FrictionRidgeGeneralizedPosition;
00057         using FGPSet = std::vector<FGP>;
00058
00072         std::ostream&
00073         operator<<(
00074             std::ostream &s,
00075             const Feature::FGP &fgp);
00076     }
00077 }
00078 #endif /* __BE_FEATURE_H__ */
00079
```

I.15 be_feature_an2k11efs.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
```

```
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FEATURE_AN2K11EFS_H__
00012 #define __BE_FEATURE_AN2K11EFS_H__
00013
00014 #include <be_image.h>
00015 #include <be_finger.h>
00016 #include <be_palm.h>
00017 #include <be_plantar.h>
00018 #include <be_feature.h>
00019 #include <be_feature_minutiae.h>
00020 #include <be_framework_enumeration.h>
00021 #include <be_memory_autoarray.h>
00022
00023 namespace BiometricEvaluation
00024 {
00025     namespace Feature {
00026     namespace AN2K11EFS {
00027
00033         struct Orientation {
00035             enum class EncodingMethod {
00037                 Default,
00042                 Indeterminate,
00044                 UserDefined
00045             };
00046
00048             static const int    EODDefault = 0;
00050             static const int    EUCDefault = 15;
00052             static const int    EUCIndeterminate = 180;
00053
00055             EncodingMethod encodingMethod;
00056
00058             int eod;
00059             bool    has_euc;
00061             int euc;
00062         };
00063         std::ostream& operator<< (std::ostream&, const Orientation&);
00064
00069         enum class FingerprintSegment {
00070             PRX = 0,
00071             DST,
00072             MED,
00073             UNK
00074         };
00075
00080         enum class OffCenterFingerPosition {
00081             T = 0,
00082             R,
00083             L
00084         };
00085         using OCF = OffCenterFingerPosition;
00086
00096         struct FPPPosition {
00098             Feature::FGP            fgp;
00099
00100            bool                has_fsm;
00102            FingerprintSegment      fsm;
00103
00104            bool                has_ocf;
00106            OffCenterFingerPosition     ocf;
00107
00108            bool                has_sgp;
00110            BiometricEvaluation::Image::CoordinateSet sgp;
00111
00112        };
00113        std::ostream& operator<< (std::ostream&, const FPPPosition&);
00114
00119        enum class TonalReversal {
00124            N = 0,
00129            P,
00133            U
00134        };
```

```
00135
00140         enum class LateralReversal {
00142             L = 0,
00144             U
00145         };
00146
00152         struct ImageInfo {
00156             BiometricEvaluation::Image::ROI  roi;
00157
00161             FPPPosition          fpp;
00162
00167             Orientation          ort;
00168
00169             bool                has_trv;
00173             TonalReversal           trv;
00174
00175             bool                has_plr;
00179             LateralReversal         plr;
00180         };
00181         std::ostream& operator<< (std::ostream&,
00182             const ImageInfo&);
00183
00189         struct MinutiaPoint : public Feature::MinutiaPoint {
00190             bool    has_mru;
00192             int mru;
00193             bool    has_mdu;
00195             int mdu;
00196         };
00197         std::ostream& operator<< (std::ostream&,
00198             const MinutiaPoint&);
00199         using MinutiaPointSet = std::vector<MinutiaPoint>;
00200
00205         struct MinutiaeRidgeCount {
00207             int mia;
00209             int mib;
00211             int mir;
00212             bool    has_mrn;
00214             int mrn;
00215             bool    has_mrs;
00217             int mrs;
00218         };
00219         std::ostream& operator<< (std::ostream&,
00220             const MinutiaeRidgeCount&);
00221         using MinutiaeRidgeCountSet =
00222             std::vector<MinutiaeRidgeCount>;
00223
00228         enum class MethodOfRidgeCounting {
00230             A = 0,
00232             T,
00234             M
00235         };
00236         using MORC = MethodOfRidgeCounting;
00237
00243         struct MinutiaeRidgeCountConfidence {
00244             Image::Coordinate  pointA;
00245             Image::Coordinate  pointB;
00246             MethodOfRidgeCounting   morc;
00247             int         mcv;
00248         };
00249         using MRCC = struct MinutiaeRidgeCountConfidence;
00250         std::ostream& operator<< (std::ostream&,
00251             const MinutiaeRidgeCountConfidence&);
00252         using MinutiaeRidgeCountConfidenceSet =
00253             std::vector<MinutiaeRidgeCountConfidence>;
00254
00259         enum class MinutiaeRidgeCountAlgorithm {
00260             OCTANT = 0,
00261             EFTS7,
00262             QUADRANT
00263         };
00264         using MRA = MinutiaeRidgeCountAlgorithm;
00265
00270         struct MinutiaeRidgeCountInfo {
00271             bool                has_mra;
00272             MinutiaeRidgeCountAlgorithm mra;
00273             bool                has_mrcs;
00274             MinutiaeRidgeCountSet       mrcs;
```

```
00275             bool                has_rccs;
00276             MinutiaeRidgeCountConfidenceSet rccs;
00277         };
00278         std::ostream& operator<< (std::ostream&,
00279             const MinutiaeRidgeCountInfo&);
00280
00281          /*
00282          * @brief
00283          * Representation of an extended feature set core.
00284          */
00285         struct CorePoint {
00286             Image::Coordinate  location;
00287             bool            has_cdi;
00288             int         cdi;
00289             bool            has_rpu;
00290             int         rpu;
00291             bool            has_duy;
00292             int         duy;
00293         };
00294         std::ostream& operator<< (std::ostream&, const CorePoint&);
00295         using CorePointSet = std::vector<CorePoint>;
00296
00301         enum class DeltaType {
00303             L,
00305             R,
00307             I00,
00309             I02,
00311             I03,
00313             I04,
00315             I05,
00317             I07,
00319             I08,
00321             I09,
00323             I10,
00325             I16,
00327             I17,
00329             C,
00331             Other
00332         };
00333
00338         struct DeltaPoint {
00339             Image::Coordinate  location;
00340             bool        has_dup;
00341             int     dup;
00342             bool        has_dlf;
00343             int     dlf;
00344             bool        has_drt;
00345             int     drt;
00346             bool        has_dtp;
00347             DeltaType   dtp;
00348             bool        has_rpu;
00349             int     rpu;
00350             bool        has_duu;
00351             int     duu;
00352             bool        has_dul;
00353             int     dul;
00354             bool        has_dur;
00355             int     dur;
00356         };
00357         using DeltaPointSet = std::vector<DeltaPoint>;
00358         std::ostream& operator<< (std::ostream&, const DeltaPoint&);
00359
00371         struct NoFeaturesPresent {
00372             bool    cores;
00373             bool    deltas;
00374             bool    minutiae;
00375         };
00376         std::ostream& operator<< (
00377              std::ostream&, const NoFeaturesPresent&);
00378
00380         enum class LatentProcessingMethod
00381         {
00382             I12,
00383             ADX,
00384             ALS,
00385             AMB,
00386             AY7,
```

```
00387             BAR,
00388             BLE,
00389             BLP,
00390             BPA,
00391             BRY,
00392             CBB,
00393             CDS,
00394             COG,
00395             DAB,
00396             DFO,
00397             FLP,
00398             GEN,
00399             GRP,
00400             GTV,
00401             HCA,
00402             IOD,
00403             ISR,
00404             LAS,
00405             LCV,
00406             LIQ,
00407             LQD,
00408             MBD,
00409             MBP,
00410             MGP,
00411             MPD,
00412             MRM,
00413             NIN,
00414             OTH,
00415             PDV,
00416             R6G,
00417             RAM,
00418             RUV,
00419             SAO,
00420             SDB,
00421             SGF,
00422             SPR,
00423             SSP,
00424             SVN,
00425             TEC,
00426             TID,
00427             VIS,
00428             WHP,
00429             ZIC
00430         };
00431         using LPM = LatentProcessingMethod;
00432
00434         enum class ValueAssessmentCode
00435         {
00436             Value,
00437             ValueForIndividualization = Value,
00438             VID = Value,
00439             Limited,
00440             ValueForExclusionOnly = Limited,
00441             VEO = Limited,
00442             NoValue,
00443             NV = NoValue,
00444             NonPrint
00445         };
00446
00448         struct ExaminerAnalysisAssessment
00449         {
00451             bool present{false};
00452
00454             ValueAssessmentCode aav;
00456             std::string aln;
00458             std::string afn;
00460             std::string aaf;
00462             std::string amt;
00464             std::string acm{};
00466             bool has_cxf{false};
00468             bool cxf{};
00469         };
00470         std::ostream&
00471         operator<<(
00472              std::ostream&,
00473              const ExaminerAnalysisAssessment&);
00474
```

```
00476         enum class SubstrateCode
00477         {
00478             Paper,
00479             Cardboard,
00480             UnfinishedWood,
00481             OtherOrUnknownPorous,
00482 
00483             Plastic,
00484             Glass,
00485             PaintedMetal,
00486             UnpaintedMetal,
00487             GlossyPaintedSurface,
00488             AdhesiveSideTape,
00489             NonAdhesiveSideTape,
00490             AluminumFoil,
00491             OtherOrUnknownNonporous,
00492 
00493             Rubber,
00494             Leather,
00495             EmulsionSidePhotograph,
00496             PaperSidePhotograph,
00497             GlossyOrSemiglossyPaperOrCardboard,
00498             SatinOrFlatFinishedPaintedSurface,
00499             OtherOrUnknownSemiporous,
00500 
00501             Other,
00502             Unknown
00503         };
00504 
00506         struct Substrate
00507         {
00509             bool present{false};
00510 
00512             SubstrateCode cls{SubstrateCode::Unknown};
00514             std::string osd{};
00515         };
00516         std::ostream&
00517         operator<<(
00518              std::ostream&,
00519              const Substrate&);
00520 
00522         struct Pattern
00523         {
00531             enum class GeneralClassification
00532             {
00533                 Arch,
00534                 Whorl,
00535                 RightSlantLoop,
00536                 LeftSlantLoop,
00537                 Amputation,
00538                 TemporarilyUnavailable,
00539                 Unclassifiable,
00540                 Scar,
00541                 DissociatedRidges
00542             };
00543 
00544             /* Details subclassification for arches */
00545             enum class ArchSubclassification
00546             {
00547                 Plain,
00548                 Tented
00549             };
00550 
00551             /* Details subclassification for whorls */
00552             enum class WhorlSubclassification
00553             {
00554                 Plain,
00555                 CentralPocketLoop,
00556                 DoubleLoop,
00557                 Accidental
00558             };
00559 
00561             enum class WhorlDeltaRelationship
00562             {
00563                 Inner,
00564                 Outer,
00565                 Meeting
```

```
00566             };
00567
00568             bool present{false};
00569
00570             GeneralClassification general;
00571
00572             bool hasSubclass{false};
00573             union
00574             {
00575                 ArchSubclassification arch;
00576                 WhorlSubclassification whorl;
00577             } subclass;
00578
00579             bool hasWhorlDeltaRelationship{false};
00580             WhorlDeltaRelationship whorlDeltaRelationship;
00581         };
00582         std::ostream&
00583         operator<<(
00584              std::ostream&,
00585              const Pattern&);
00586
00596         class ExtendedFeatureSet {
00597         public:
00623             ExtendedFeatureSet(
00624                 const std::string &filename,
00625                 int recordNumber);
00626
00647             ExtendedFeatureSet(
00648                 Memory::uint8Array &buf,
00649                 int recordNumber);
00650
00659             ImageInfo
00660             getImageInfo()
00661                 const;
00662
00673             BiometricEvaluation::Feature::AN2K11EFS::MinutiaPointSet
00674             getMPS()
00675                 const;
00676
00689             BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCountInfo
00690             getMRCI()
00691                 const;
00692
00703             BiometricEvaluation::Feature::AN2K11EFS::CorePointSet
00704             getCPS()
00705                 const;
00706
00717             BiometricEvaluation::Feature::AN2K11EFS::DeltaPointSet
00718             getDPS()
00719                 const;
00720
00731             std::vector<LatentProcessingMethod>
00732             getLPM()
00733                 const;
00734
00740             BiometricEvaluation::Feature::AN2K11EFS::NoFeaturesPresent
00741             getNFP()
00742                 const;
00743
00752             ExaminerAnalysisAssessment
00753             getEAA()
00754                 const;
00755
00761             Substrate
00762             getLSB()
00763                 const;
00764
00769             std::vector<Pattern>
00770             getPAT()
00771                 const;
00772
00773             ~ExtendedFeatureSet();
00774
00775         private:
00776             class Impl;
00777             std::unique_ptr<ExtendedFeatureSet::Impl> pimpl;
00778         };
```

```
00779     } /* Namespace AN2K11EFS */
00780     } /* Namespace Feature */
00781 }
00782
00783 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00784     BiometricEvaluation::Feature::AN2K11EFS::Orientation::EncodingMethod,
00785     BE_Feature_AN2K11EFS_Orientation_EncodingMethod_EnumToStringMap);
00786
00787 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00788     BiometricEvaluation::Feature::AN2K11EFS::FingerprintSegment,
00789     BE_Feature_AN2K11EFS_FingerprintSegment_EnumToStringMap);
00790
00791 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00792     BiometricEvaluation::Feature::AN2K11EFS::OCF,
00793     BE_Feature_AN2K11EFS_OCF_EnumToStringMap);
00794
00795 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00796     BiometricEvaluation::Feature::AN2K11EFS::TonalReversal,
00797     BE_Feature_AN2K11EFS_TonalReversal_EnumToStringMap);
00798
00799 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00800     BiometricEvaluation::Feature::AN2K11EFS::LateralReversal,
00801     BE_Feature_AN2K11EFS_LateralReversal_EnumToStringMap);
00802
00803 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00804     BiometricEvaluation::Feature::AN2K11EFS::MethodOfRidgeCounting,
00805     BE_Feature_AN2K11EFS_MethodOfRidgeCounting_EnumToStringMap);
00806
00807 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00808     BiometricEvaluation::Feature::AN2K11EFS::MRA,
00809     BE_Feature_AN2K11EFS_MRA_EnumToStringMap);
00810
00811 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00812     BiometricEvaluation::Feature::AN2K11EFS::DeltaType,
00813     BE_Feature_AN2K11EFS_DeltaType_EnumToStringMap);
00814
00815 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00816     BiometricEvaluation::Feature::AN2K11EFS::LatentProcessingMethod,
00817     BE_Feature_AN2K11EFS_LatentProcessingMethod_EnumToStringMap);
00818
00819 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00820     BiometricEvaluation::Feature::AN2K11EFS::ValueAssessmentCode,
00821     BE_Feature_AN2K11EFS_ValueAssessmentCode_EnumToStringMap);
00822
00823 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00824     BiometricEvaluation::Feature::AN2K11EFS::SubstrateCode,
00825     BE_Feature_AN2K11EFS_SubstrateCode_EnumToStringMap);
00826
00827 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00828     BiometricEvaluation::Feature::AN2K11EFS::Pattern::GeneralClassification,
00829     BE_Feature_AN2K11EFS_Pattern_GeneralClassification_EnumToStringMap);
00830
00831 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00832     BiometricEvaluation::Feature::AN2K11EFS::Pattern::ArchSubclassification,
00833     BE_Feature_AN2K11EFS_Pattern_ArchSubclassification_EnumToStringMap);
00834
00835 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00836     BiometricEvaluation::Feature::AN2K11EFS::Pattern::WhorlSubclassification,
00837     BE_Feature_AN2K11EFS_Pattern_WhorlSubclassification_EnumToStringMap);
00838
00839 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00840     BiometricEvaluation::Feature::AN2K11EFS::Pattern::WhorlDeltaRelationship,
00841     BE_Feature_AN2K11EFS_Pattern_WhorlDeltaRelationship_EnumToStringMap);
00842
00843 #endif /* __BE_FEATURE_AN2K11EFS_H__ */
00844
```

I.16 be_feature_an2k7minutiae.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
```

```
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FEATURE_AN2K7MINUTIAE_H__
00012 #define __BE_FEATURE_AN2K7MINUTIAE_H__
00013
00014 #include <iostream>
00015
00016 #include <be_framework_enumeration.h>
00017 #include <be_feature_minutiae.h>
00018 #include <be_finger.h>
00019 #include <be_memory_autoarray.h>
00020
00021 namespace BiometricEvaluation
00022 {
00023     namespace Feature
00024     {
00033         class AN2K7Minutiae : public Minutiae {
00034         public:
00035
00040             class PatternClassification {
00041             public:
00042                 struct Entry {
00054                      Entry(
00055                         bool standard,
00056                         std::string code);
00057
00062                     bool standard;
00067                     std::string code;
00068                 };
00069                 using Entry = struct Entry;
00070
00071                 PatternClassification() = delete;
00072             };
00073             using PatternClassificationSet =
00074                 std::vector<PatternClassification::Entry>;
00075
00088             static Finger::PatternClassification
00089             convertPatternClassification(
00090                 const char *fpc);
00091
00103             static Finger::PatternClassification
00104             convertPatternClassification(
00105                 const PatternClassification::Entry &entry);
00106
00111             enum class EncodingMethod
00112             {
00114                 Automatic = 0,
00116                 AutomaticUnedited,
00118                 AutomaticEdited,
00119                 /* Manually encoded */
00120                 Manual
00121             };
00122
00135             static EncodingMethod
00136             convertEncodingMethod(
00137                 const char *mem);
00138
00144             struct FingerprintReadingSystem {
00146                 std::string name;
00148                 EncodingMethod method;
00150                 std::string equipment;
00151             };
00152             using FingerprintReadingSystem =
00153                 struct FingerprintReadingSystem;
00154
00178             AN2K7Minutiae(
00179                 const std::string &filename,
00180                 int recordNumber);
00181
00202             AN2K7Minutiae(
00203                 Memory::uint8Array &buf,
00204                 int recordNumber);
00205
00214             PatternClassificationSet
00215                 getPatternClassificationSet() const;
```

```
00216
00224             FingerprintReadingSystem
00225             getOriginatingFingerprintReadingSystem() const;
00226
00228             Finger::PositionSet
00229             getPositions()
00230                 const;
00231
00250             static Image::Coordinate
00251             convertCoordinate(
00252                 const char *str,
00253                 bool calculateDistance = true);
00254
00255             /*
00256              * Feature::Minutiae implementations.
00257              */
00258             MinutiaeFormat getFormat() const;
00259             MinutiaPointSet getMinutiaPoints() const;
00260             RidgeCountItemSet getRidgeCountItems() const;
00261             CorePointSet getCores() const;
00262             DeltaPointSet getDeltas() const;
00263
00264         protected:
00265         private:
00266             void readType9Record(
00267                 Memory::uint8Array &buf,
00268                     int recordNumber);
00269
00270             MinutiaPointSet _minutiaPointSet;
00271             RidgeCountItemSet _ridgeCountItemSet;
00272             CorePointSet _corePointSet;
00273             DeltaPointSet _deltaPointSet;
00274             FingerprintReadingSystem _ofr;
00275             PatternClassificationSet _fpc;
00276             Finger::PositionSet _fgp;
00277             std::string _userdefinedFpc;
00278
00279         };
00280         using AN2K7MinutiaeSet =
00281             std::vector<std::shared_ptr<AN2K7Minutiae>>;
00282
00287         std::ostream&
00288         operator<<(
00289             std::ostream&,
00290             const AN2K7Minutiae::FingerprintReadingSystem&);
00291     }
00292 }
00293
00294 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00295     BiometricEvaluation::Feature::AN2K7Minutiae::EncodingMethod,
00296     BE_Feature_EncodingMethod_EnumToStringMap);
00297
00298 #endif /* __BE_FEATURE_AN2K7MINUTIAE_H__ */
00299
```

I.17 be_feature_incitsminutiae.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FEATURE_INCITSMINUTIAE_H__
00012 #define __BE_FEATURE_INCITSMINUTIAE_H__
00013
00014 #include <iostream>
00015
00016 #include <be_feature_minutiae.h>
00017 #include <be_memory_autoarray.h>
00018 #include <be_memory_indexedbuffer.h>
```

```
00019
00020 namespace BiometricEvaluation
00021 {
00022     namespace Feature
00023     {
00037         class INCITSMinutiae : public Minutiae {
00038         public:
00039
00040             /*
00041              * Constants relevant to INCITS and ISO finger minutiae
00042              * data records.
00043              */
00044             static const std::string FMR_ANSI_SPEC_VERSION;
00045             static const std::string FMR_ISO_SPEC_VERSION;
00046             static const std::string FMR_ANSI07_SPEC_VERSION;
00047             static const uint8_t FMR_SPEC_VERSION_LEN = 4;
00048
00049             /*
00050              * Define the lengths of data blocks in the finger
00051              * minutiae record.
00052              */
00053             static const uint32_t FED_HEADER_LENGTH = 4;
00054             static const uint32_t FED_RCD_ITEM_LENGTH = 3;
00055
00056             /*
00057              * Define the masks for the minutia type and x/y
00058              * coordinates within a minutia data record
00059              */
00060             static const uint16_t FMD_MINUTIA_TYPE_MASK = 0xC000;
00061             static const uint16_t FMD_RESERVED_MASK = 0xC000;
00062             static const uint16_t FMD_MINUTIA_TYPE_SHIFT = 14;
00063             static const uint16_t FMD_RESERVED_SHIFT = 14;
00064             static const uint16_t FMD_X_COORD_MASK = 0x3FFF;
00065             static const uint16_t FMD_Y_COORD_MASK = 0x3FFF;
00066
00067             /*
00068              * The ISO Compact FMD record has type encoded with the
00069              * angle value.
00070              */
00071             static const uint16_t FMD_ISO_COMPACT_MINUTIA_TYPE_MASK
00072                 = 0xC0;
00073             static const uint16_t FMD_ISO_COMPACT_MINUTIA_TYPE_SHIFT
00074                 = 6;
00075             static const uint16_t FMD_ISO_COMPACT_MINUTIA_ANGLE_MASK
00076                 = 0x3F;
00077
00078             /* Range of the Minutia Quality values */
00079             static const uint16_t FMD_MIN_MINUTIA_QUALITY = 0;
00080             static const uint16_t FMD_MAX_MINUTIA_QUALITY = 100;
00081             static const uint16_t FMD_UNKNOWN_MINUTIA_QUALITY = 0;
00082
00083             /* Range of Minutia Angle values. */
00084             static const uint16_t FMD_MIN_MINUTIA_ANGLE = 0;
00085             static const uint16_t FMD_MAX_MINUTIA_ANGLE = 179;
00086             static const uint16_t FMD_MAX_MINUTIA_ISONC_ANGLE = 255;
00087             static const uint16_t FMD_MAX_MINUTIA_ISOCC_ANGLE = 63;
00088
00089             /*
00090              * What each unit of angle represents in terms of
00091              * degrees.
00092              */
00093             static const uint16_t FMD_ANSI_ANGLE_UNIT = 2;
00094             static const uint16_t FMD_ISO_ANGLE_UNIT;
00095             static const uint16_t FMD_ISOCC_ANGLE_UNIT;
00096
00097             /* Types of Minutia */
00098             static const uint16_t FMD_MINUTIA_TYPE_OTHER = 0;
00099             static const uint16_t FMD_MINUTIA_TYPE_RIDGE_ENDING = 1;
00100             static const uint16_t FMD_MINUTIA_TYPE_BIFURCATION = 2;
00101
00102             /* Range of the Finger Quality values */
00103             static const uint16_t FMR_MIN_FINGER_QUALITY = 0;
00104             static const uint16_t FMR_MAX_FINGER_QUALITY = 100;
00105             static const uint16_t ISO_UNKNOWN_FINGER_QUALITY = 0;
00106
00107             /* Extended Data Area Type Codes */
00108             static const uint16_t FED_RESERVED = 0x0000;
```

```
00109             static const uint16_t FED_RIDGE_COUNT = 0x0001;
00110             static const uint16_t FED_CORE_AND_DELTA = 0x0002;
00111 
00112             /* Ridge Count Extraction Method Codes */
00113             static const uint16_t RCE_NONSPECIFIC = 0x00;
00114             static const uint16_t RCE_FOUR_NEIGHBOR = 0x01;
00115             static const uint16_t RCE_EIGHT_NEIGHBOR = 0x02;
00116 
00117             /* Core and Delta type codes. */
00118             static const uint16_t CORE_TYPE_NONANGULAR = 0x00;
00119             static const uint16_t CORE_TYPE_ANGULAR = 0x01;
00120             static const uint16_t DELTA_TYPE_NONANGULAR = 0x00;
00121             static const uint16_t DELTA_TYPE_ANGULAR = 0x01;
00122 
00123 
00124             /*
00125              * Feature::Minutiae implementations.
00126              */
00127             MinutiaeFormat getFormat() const;
00128             MinutiaPointSet getMinutiaPoints() const;
00129             RidgeCountItemSet getRidgeCountItems() const;
00130             CorePointSet getCores() const;
00131             DeltaPointSet getDeltas() const;
00132 
00151             INCITSMinutiae(
00152                 const MinutiaPointSet &mps,
00153                 const RidgeCountItemSet &rcis,
00154                 const CorePointSet &cps,
00155                 const DeltaPointSet &dps);
00156 
00161             INCITSMinutiae();
00162 
00168              void setMinutiaPoints(
00169                   const MinutiaPointSet& mps);
00170 
00177             void setRidgeCountItems(
00178                 const RidgeCountItemSet& rcis);
00179 
00186             void setCorePointSet(
00187                 const CorePointSet& cps);
00188 
00195             void setDeltaPointSet(
00196                 const DeltaPointSet& dps);
00197 
00198         private:
00199             MinutiaPointSet _minutiaPointSet;
00200             RidgeCountItemSet _ridgeCountItemSet;
00201             CorePointSet _corePointSet;
00202             DeltaPointSet _deltaPointSet;
00203 
00204         };
00205 
00206     }
00207 }
00208 
00209 #endif /* __BE_FEATURE_INCITSMINUTIAE_H__ */
00210 
```

I.18 be_feature_minutiae.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010 
00011 #ifndef __BE_FEATURE_MINUTIAE_H__
00012 #define __BE_FEATURE_MINUTIAE_H__
00013 
00014 #include <iostream>
00015 #include <memory>
```

```
00016 #include <string>
00017 #include <vector>
00018
00019 #include <be_error.h>
00020 #include <be_finger.h>
00021 #include <be_framework_enumeration.h>
00022 #include <be_image.h>
00023 #include <be_memory_autoarray.h>
00024
00025 namespace BiometricEvaluation
00026 {
00027     namespace Feature
00028     {
00033         enum class MinutiaeFormat
00034         {
00035             AN2K7 = 0,
00036             IAFIS,
00037             Cogent,
00038             Motorola,
00039             Sagem,
00040             NEC,
00041             Identix,
00042             M1,
00043             Other
00044         };
00045
00051         enum class MinutiaeType
00052         {
00053             RidgeEnding = 0,
00054             Bifurcation,
00055             Compound,
00056             NoDistinction,
00057             Other
00058         };
00059
00064         struct MinutiaPoint
00065         {
00066             unsigned int        index;
00067             bool            has_type;
00068             MinutiaeType        type;
00069             Image::Coordinate  coordinate;
00070             unsigned int        theta;
00071             bool            has_quality;
00072             unsigned int        quality;
00073         };
00074         using MinutiaPoint = struct MinutiaPoint;
00075         std::ostream& operator<< (std::ostream&,
00076             const MinutiaPoint&);
00077         using MinutiaPointSet = std::vector<MinutiaPoint>;
00078
00083         enum class RidgeCountExtractionMethod
00084         {
00085             NonSpecific = 0,
00086             FourNeighbor = 1,
00087             EightNeighbor = 2,
00088             Other = 3
00089         };
00090
00097         struct RidgeCountItem {
00098
00103             RidgeCountItem(
00104                 RidgeCountExtractionMethod extraction_method,
00105                 int index_one,
00106                 int index_two,
00107                 int count = 0);
00108             RidgeCountItem() { }
00109             RidgeCountExtractionMethod extraction_method;
00110             int     index_one;
00111             int     index_two;
00112             int     count;
00113         };
00114         using RidgeCountItem = struct RidgeCountItem;
00115         std::ostream& operator<< (std::ostream&,
00116             const RidgeCountItem&);
00117         using RidgeCountItemSet = std::vector<RidgeCountItem>;
00118
00127         struct CorePoint {
```

```
00132             CorePoint(
00133                 Image::Coordinate coordinate,
00134                 bool has_angle = false,
00135                 int angle = 0);
00136 
00137             Image::Coordinate  coordinate;
00138             bool            has_angle;
00139             int         angle;
00140         };
00141         using CorePoint = struct CorePoint;
00142         std::ostream& operator<< (std::ostream&,const CorePoint&);
00143         using CorePointSet = std::vector<CorePoint>;
00144 
00153         struct DeltaPoint {
00158             DeltaPoint(
00159                 Image::Coordinate coordinate,
00160                 bool has_angle = false,
00161                 int angle1 = 0,
00162                 int angle2 = 0,
00163                 int angle3 = 0);
00164 
00165             Image::Coordinate  coordinate;
00166             bool            has_angle;
00167             int         angle1;
00168             int         angle2;
00169             int         angle3;
00170         };
00171         using DeltaPoint = struct DeltaPoint;
00172         std::ostream& operator<< (std::ostream&,const DeltaPoint&);
00173 
00174         using DeltaPointSet = std::vector<DeltaPoint>;
00175 
00186         class Minutiae {
00187         public:
00188             virtual ~Minutiae();
00189 
00195             virtual MinutiaeFormat getFormat() const = 0;
00196 
00202             virtual MinutiaPointSet getMinutiaPoints() const = 0;
00203 
00209             virtual RidgeCountItemSet getRidgeCountItems()
00210                 const = 0;
00211 
00217             virtual CorePointSet getCores() const = 0;
00218 
00224             virtual DeltaPointSet getDeltas() const = 0;
00225 
00226         protected:
00227             Minutiae() {};
00228 
00229         private:
00230         };
00231         using MinutiaeSet = std::vector<std::shared_ptr<Minutiae>>;
00232     }
00233 }
00234 #endif /* __BE_FEATURE_MINUTIAE_H__ */
00235 
00236 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00237     BiometricEvaluation::Feature::MinutiaeFormat,
00238     BE_Feature_MinutiaeFormat_EnumToStringMap);
00239 
00240 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00241     BiometricEvaluation::Feature::MinutiaeType,
00242     BE_Feature_MinutiaeType_EnumToStringMap);
00243 
00244 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00245     BiometricEvaluation::Feature::RidgeCountExtractionMethod,
00246     BE_Feature_RidgeCountExtractionMethod_EnumToStringMap);
```

I.19 be_feature_mpegfacepoint.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
```

```
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FEATURE__MPEGFACEPOINT_H__
00012 #define __BE_FEATURE__MPEGFACEPOINT_H__
00013
00014 #include <vector>
00015 #include <be_image.h>
00016
00017 namespace BiometricEvaluation
00018 {
00023     namespace Feature
00024     {
00029         typedef struct {
00030             uint8_t     type;
00031             uint8_t     major;
00032             uint8_t     minor;
00033             BiometricEvaluation::Image::Coordinate coordinate;
00034         } MPEGFacePoint;
00035         typedef std::vector<MPEGFacePoint> MPEGFacePointSet;
00036     }
00037 }
00038 #endif /* __BE_FEATURE__MPEGFACEPOINT_H__ */
00039
```

I.20 be_feature_sort.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties.  Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain.  NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_FEATURE_SORT_H_
00012 #define BE_FEATURE_SORT_H_
00013
00014 #include <be_feature_minutiae.h>
00015 #include <be_image.h>
00016
00017 namespace BiometricEvaluation
00018 {
00019     namespace Feature
00020     {
00022         namespace Sort
00023         {
00025             enum class Kind
00026             {
00032                 XYAscending,
00038                 XYDescending,
00044                 YXAscending,
00050                 YXDescending,
00052                 QualityAscending,
00054                 QualityDescending,
00056                 AngleAscending,
00058                 AngleDescending,
00064                 PolarCOMAscending,
00070                 PolarCOMDescending,
00076                 PolarCOIAscending,
00082                 PolarCOIDescending,
00084                 Unknown
00085             };
00086
00088             class XY
00089             {
00090             public:
00096                 bool
00097                 operator()(
00098                     const BiometricEvaluation::Feature::
```

```
00099                   MinutiaPoint &lhs,
00100                   const BiometricEvaluation::Feature::
00101                   MinutiaPoint &rhs)
00102                   const;
00103           };
00104
00106           class YX
00107           {
00108           public:
00114               bool
00115               operator()(
00116                   const BiometricEvaluation::Feature::
00117                   MinutiaPoint &lhs,
00118                   const BiometricEvaluation::Feature::
00119                   MinutiaPoint &rhs)
00120                   const;
00121           };
00122
00124           class Quality
00125           {
00126           public:
00131               bool
00132               operator()(
00133                   const BiometricEvaluation::Feature::
00134                   MinutiaPoint &lhs,
00135                   const BiometricEvaluation::Feature::
00136                   MinutiaPoint &rhs)
00137                   const;
00138           };
00139
00141           class Angle
00142           {
00143           public:
00145               bool
00146               operator()(
00147                   const BiometricEvaluation::Feature::
00148                   MinutiaPoint &lhs,
00149                   const BiometricEvaluation::Feature::
00150                   MinutiaPoint &rhs)
00151                   const;
00152           };
00153
00159           class Polar
00160           {
00161           public:
00172               Polar(
00173                   const BiometricEvaluation::Image::Coordinate
00174                   &center);
00175
00177               bool
00178               operator()(
00179                   const BiometricEvaluation::Feature::
00180                   MinutiaPoint &lhs,
00181                   const BiometricEvaluation::Feature::
00182                   MinutiaPoint &rhs)
00183                   const;
00184
00198               static BiometricEvaluation::Image::Coordinate
00199               centerOfMinutiaeMass(
00200                   const BiometricEvaluation::Feature::
00201                   MinutiaPointSet &mps);
00202
00214               static BiometricEvaluation::Image::Coordinate
00215               centerOfImage(
00216                   const BiometricEvaluation::Image::Size
00217                   &size);
00218
00219           private:
00221               BiometricEvaluation::Image::Coordinate _center;
00222
00237               uint64_t
00238               distanceFromCenter(
00239                   const BiometricEvaluation::Image::Coordinate
00240                   &coordinate)
00241                   const;
00242           };
00243
```

```
00249             void
00250             updateIndicies(
00251                 BiometricEvaluation::Feature::MinutiaPointSet &mps);
00252
00267             std::vector<Feature::MinutiaPoint>
00268             sort(
00269                 std::vector<Feature::MinutiaPoint> &minutia,
00270                 const Kind &sortOrder);
00271
00287             std::vector<Feature::MinutiaPoint>
00288             stableSort(
00289                 std::vector<Feature::MinutiaPoint> &minutia,
00290                 const Kind &sortOrder);
00291         }
00292     }
00293 }
00294
00295 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00296     BiometricEvaluation::Feature::Sort::Kind,
00297     BE_Feature_Sort_Kind_EnumToStringMap);
00298
00299 #endif /* BE_FEATURE_SORT_H_ */
```

I.21 be_finger.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FINGER_H__
00012 #define __BE_FINGER_H__
00013
00014 #include <iostream>
00015 #include <map>
00016 #include <vector>
00017
00018 #include <be_framework_enumeration.h>
00019
00020 namespace BiometricEvaluation
00021 {
00033     namespace Finger
00034     {
00036         enum class PatternClassification
00037         {
00038             PlainArch = 0,
00039             TentedArch,
00040             RadialLoop,
00041             UlnarLoop,
00042             PlainWhorl,
00043             CentralPocketLoop,
00044             DoubleLoop,
00045             AccidentalWhorl,
00046             Whorl,
00047             RightSlantLoop,
00048             LeftSlantLoop,
00049             Scar,
00050             Amputation,
00051             Unknown
00052         };
00053
00061         enum class Position
00062         {
00063             Unknown         = 0,
00064             RightThumb      = 1,
00065             RightIndex      = 2,
00066             RightMiddle     = 3,
00067             RightRing       = 4,
00068             RightLittle     = 5,
00069             LeftThumb       = 6,
```

```
00070             LeftIndex       = 7,
00071             LeftMiddle      = 8,
00072             LeftRing        = 9,
00073             LeftLittle      = 10,
00074             PlainRightThumb     = 11,
00075             PlainLeftThumb      = 12,
00076             PlainRightFourFingers   = 13,
00077             PlainLeftFourFingers    = 14,
00078             LeftRightThumbs     = 15,
00079             RightExtraDigit     = 16,
00080             LeftExtraDigit      = 17,
00081             UnknownFrictionRidge    = 18,
00082             EJI         = 19,
00083             RightIndexMiddle    = 40,
00084             RightMiddleRing     = 41,
00085             RightRingLittle     = 42,
00086             LeftIndexMiddle     = 43,
00087             LeftMiddleRing      = 44,
00088             LeftRingLittle      = 45,
00089             RightIndexLeftIndex = 46,
00090             RightIndexMiddleRing    = 47,
00091             RightMiddleRingLittle   = 48,
00092             LeftIndexMiddleRing = 49,
00093             LeftMiddleRingLittle    = 50,
00094             PlainRightFourTips  = 51,
00095             PlainLeftFourTips   = 52,
00096             PlainRightFiveTips  = 53,
00097             PlainLeftFiveTips   = 54,
00098         };
00099         using PositionSet = std::vector<Position>;
00100
00102         enum class Impression
00103         {
00104             PlainContact                = 0,
00105             LiveScanPlain               = 0,
00106             RolledContact               = 1,
00107             LiveScanRolled              = 1,
00108             NonLiveScanPlain            = 2,
00109             NonLiveScanRolled           = 3,
00110             LatentImage             = 4,
00111             LatentImpression            = 4,
00112             LatentTracing               = 5,
00113             LatentPhoto             = 6,
00114             LatentLift              = 7,
00115             LiveScanSwipe               = 8,
00116             LiveScanVerticalSwipe           = 8,
00117             LiveScanPalm                = 10,
00118             NonLiveScanPalm             = 11,
00119             LatentPalmImpression            = 12,
00120             LatentPalmTracing           = 13,
00121             LatentPalmPhoto             = 14,
00122             LatentPalmLift              = 15,
00123             LiveScanOpticalContactPlain     = 20,
00124             LiveScanOpticalContactRolled        = 21,
00125             LiveScanNonOpticalContactPlain      = 22,
00126             LiveScanNonOpticalContactRolled     = 23,
00127             ContactlessPlainStationarySubject   = 24,
00128             LiveScanOpticalContactlessPlain     = 24,
00129             ContactlessRolledStationarySubject  = 25,
00130             LiveScanOpticalContactlessRolled    = 25,
00131             LiveScanNonOpticalContactlessPlain  = 26,
00132             LiveScanNonOpticalContactlessRolled = 27,
00133             Other                   = 28,
00134             Unknown                 = 29,
00135             ContactlessRolledMovingSubject      = 41,
00136             ContactlessPlainMovingSubject       = 42
00137         };
00138
00140         enum class FingerImageCode {
00141             EJI = 0,
00142             RolledTip,
00143             FullFingerRolled,
00144             FullFingerPlainLeft,
00145             FullFingerPlainCenter,
00146             FullFingerPlainRight,
00147             ProximalSegment,
00148             DistalSegment,
```

```
00149             MedialSegment,
00150             NA
00151         };
00152         using PositionDescriptors = std::map<Position, FingerImageCode>;
00153
00155         enum class CaptureTechnology
00156         {
00157             Unknown                 = 0,
00158             Other                   = 1,
00159             ScannedInkOnPaper           = 2,
00160             OpticalTIRBright            = 3,
00161             OpticalTIRDark              = 4,
00162             OpticalDINative             = 5,
00163             OpticalDILowFrequenceyUnwrapped     = 6,
00164             ThreeDimensionalHighFrequencyUnwrapped  = 7,
00165             Capacitive              = 9,
00166             CapacitiveRF                = 10,
00167             Electroluminescent          = 11,
00168             ReflectedUltrasonic         = 12,
00169             UltrasonicImpediography         = 13,
00170             Thermal                 = 14,
00171             DirectPressureSensitive         = 15,
00172             IndirectPressure            = 16,
00173             LiveTape                = 17,
00174             LatentImpression            = 18,
00175             LatentPhoto             = 19,
00176             LatentMold              = 20,
00177             LatentTracing               = 21,
00178             LatentLift              = 22
00179         };
00180     }
00181 }
00182
00183 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00184     BiometricEvaluation::Finger::PatternClassification,
00185     BE_Finger_PatternClassification_EnumToStringMap);
00186
00187 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00188     BiometricEvaluation::Finger::Position,
00189     BE_Finger_Position_EnumToStringMap);
00190
00191  BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00192     BiometricEvaluation::Finger::Impression,
00193     BE_Finger_Impression_EnumToStringMap);
00194
00195 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00196     BiometricEvaluation::Finger::FingerImageCode,
00197     BE_Finger_FingerImageCode_EnumToStringMap);
00198
00199 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00200     BiometricEvaluation::Finger::CaptureTechnology,
00201     BE_Finger_CaptureTechnology_EnumToStringMap);
00202
00203 #endif /* __BE_FINGER_H__ */
00204
```

I.22 be_finger_an2kminutiae_data_record.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FINGER_AN2KMINUTIAE_DATA_RECORD_H__
00012 #define __BE_FINGER_AN2KMINUTIAE_DATA_RECORD_H__
00013
00014 #include <map>
00015 #include <memory>
00016
00017 #include <be_feature_an2k7minutiae.h>
```

```
00018 #include <be_feature_an2k11efs.h>
00019 #include <be_memory_autoarray.h>
00020
00021 /* an2k.h forward declares */
00022 struct record;
00023 typedef record RECORD;
00024
00025 namespace BiometricEvaluation {
00026     namespace Finger {
00038         class AN2KMinutiaeDataRecord {
00039         public:
00064             AN2KMinutiaeDataRecord(
00065                 const std::string &filename,
00066                 int recordNumber);
00067
00088             AN2KMinutiaeDataRecord(
00089                 Memory::uint8Array &buf,
00090                 int recordNumber);
00091
00102             std::shared_ptr<Feature::AN2K7Minutiae>
00103             getAN2K7Minutiae()
00104                 const;
00105
00117             std::shared_ptr<Feature::AN2K11EFS::ExtendedFeatureSet>
00118             getAN2K11EFS()
00119                 const;
00120
00129             Impression
00130             getImpressionType()
00131                 const;
00132
00156             std::map<uint16_t, Memory::uint8Array>
00157             getRegisteredVendorBlock(
00158                 Feature::MinutiaeFormat vendor) const;
00159
00161             int
00162             getIDC()
00163                 const;
00164
00165         protected:
00166
00167         private:
00183             void
00184             readType9Record(
00185                 Memory::uint8Array &buf,
00186                 int recordNumber);
00187
00204             void
00205             readRegisteredVendorBlock(
00206                 RECORD *type9,
00207                 Feature::MinutiaeFormat vendor);
00208
00210             std::shared_ptr<Feature::AN2K7Minutiae> _AN2K7Features;
00212             std::map<uint16_t, Memory::uint8Array> _IAFISFeatures;
00214             std::map<uint16_t, Memory::uint8Array> _cogentFeatures;
00216             std::map<uint16_t, Memory::uint8Array> _motorolaFeatures;
00218             std::shared_ptr<Feature::AN2K11EFS::ExtendedFeatureSet>
00219                 _AN2K11EFS;
00221             std::map<uint16_t, Memory::uint8Array> _sagemFeatures;
00223             std::map<uint16_t, Memory::uint8Array> _NECFeatures;
00225             std::map<uint16_t, Memory::uint8Array> _M1Features;
00227             std::map<uint16_t, Memory::uint8Array> _identixFeatures;
00229             std::map<uint16_t, Memory::uint8Array> _otherFeatures;
00230
00231
00233             int _idc{};
00235             Impression _imp;
00236         };
00237     }
00238 }
00239
00240 #endif /* __BE_FINGER_AN2KMINUTIAE_DATA_RECORD_H__ */
```

I.23 be_finger_an2kview.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FINGER_AN2KVIEW_H__
00012 #define __BE_FINGER_AN2KVIEW_H__
00013
00014 #include <be_view_an2kview.h>
00015 #include <be_feature_an2k7minutiae.h>
00016 #include <be_finger_an2kminutiae_data_record.h>
00017
00018 /* an2k.h forward declares */
00019 struct field;
00020 typedef field FIELD;
00021
00022 namespace BiometricEvaluation
00023 {
00024     namespace Finger
00025     {
00026
00043         class AN2KView : public View::AN2KView {
00044         public:
00056             static Finger::Position
00057                 convertPosition(int an2kFGP);
00058
00068             static Finger::PositionSet populateFGP(FIELD* field);
00069
00074             static Finger::Impression
00075                 convertImpression(const unsigned char *str);
00076
00087                 static Finger::FingerImageCode
00088             convertFingerImageCode(
00089                 const char *str);
00090
00102             std::vector<AN2KMinutiaeDataRecord>
00103             getMinutiaeDataRecordSet() const;
00104
00115             Finger::PositionSet getPositions() const;
00116
00123             Finger::Impression getImpressionType() const;
00124
00125         protected:
00126
00150             AN2KView(
00151                 const std::string filename,
00152                 const RecordType typeID,
00153                 const uint32_t recordNumber);
00154
00175             AN2KView(
00176                 Memory::uint8Array &buf,
00177                 const RecordType typeID,
00178                 const uint32_t recordNumber);
00179
00188             void
00189             addMinutiaeDataRecord(
00190                 Finger::AN2KMinutiaeDataRecord &mdr);
00191
00199             void setPositions(Finger::PositionSet &ps);
00200
00207             void setImpressionType(Finger::Impression &imp);
00208
00209         private:
00219             void readImageRecord(
00220                 const RecordType typeID,
00221                 const uint32_t recordNumber);
00222
00223             Finger::PositionSet _positions;
00224             std::vector<Finger::AN2KMinutiaeDataRecord>
00225                 _minutiaeDataRecordSet;
```

```
00226             Finger::Impression _imp;
00227         };
00228     }
00229 }
00230 #endif /* __BE_FINGER_AN2KVIEW_H__ */
00231
```

I.24 be_finger_an2kview_capture.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FINGER_AN2KVIEW_CAPTURE_H__
00012 #define __BE_FINGER_AN2KVIEW_CAPTURE_H__
00013
00014 #include <be_view_an2kview_varres.h>
00015 #include <be_framework_enumeration.h>
00016
00017 namespace BiometricEvaluation
00018 {
00019     namespace Finger
00020     {
00029         class AN2KViewCapture : public View::AN2KViewVariableResolution {
00030         public:
00036             enum class AmputatedBandaged
00037             {
00039                 Amputated,
00041                 Bandaged,
00043                 NA
00044             };
00045
00050             struct FingerSegmentPosition {
00061                 FingerSegmentPosition(
00062                     const Finger::Position fingerPosition,
00063                     const Image::CoordinateSet coordinates);
00064
00066                 Finger::Position fingerPosition;
00068                 Image::CoordinateSet coordinates;
00069             };
00070             using FingerSegmentPosition =
00071                 struct FingerSegmentPosition;
00072             using FingerSegmentPositionSet =
00073                 std::vector<FingerSegmentPosition>;
00074
00097             AN2KViewCapture(
00098                 const std::string &filename,
00099                 const uint32_t recordNumber);
00100
00109             AN2KViewCapture(
00110                 Memory::uint8Array &buf,
00111                 const uint32_t recordNumber);
00112
00127             QualityMetricSet
00128             extractNISTQuality(
00129                 const FIELD *field);
00130
00140             Finger::Position getPosition() const;
00141
00146             PositionDescriptors
00147             getPrintPositionDescriptors()
00148                 const;
00149
00157             PrintPositionCoordinateSet
00158             getPrintPositionCoordinates()
00159                 const;
00160
00174             QualityMetricSet
00175             getNISTQualityMetric()
```

```
00176                 const;
00177
00187             QualityMetricSet
00188             getSegmentationQualityMetric()
00189                 const;
00190
00195             AmputatedBandaged
00196             getAmputatedBandaged()
00197                 const;
00198
00204             FingerSegmentPositionSet
00205             getFingerSegmentPositionSet()
00206                 const;
00207
00213             FingerSegmentPositionSet
00214             getAlternateFingerSegmentPositionSet()
00215                 const;
00216
00225             QualityMetricSet
00226             getFingerprintQualityMetric()
00227                 const;
00228
00229         protected:
00230         private:
00232             FingerSegmentPositionSet _afsps;
00234             AmputatedBandaged _amp;
00236             DeviceMonitoringMode _dmm;
00238             FingerSegmentPositionSet _fsps;
00240             QualityMetricSet _nqm;
00242             QualityMetricSet _sqm;
00243
00244             void readImageRecord();
00245         };
00246
00251         std::ostream&
00252         operator<<(
00253             std::ostream &stream,
00254             const AN2KViewCapture::FingerSegmentPosition &fsp);
00255     }
00256 }
00257
00258 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00259     BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged,
00260     BE_Finger_AN2KViewCapture_AmputatedBandaged_EnumToStringMap);
00261
00262 #endif /* __BE_FINGER_AN2KVIEW_CAPTURE_H__ */
00263
```

I.25 be_finger_an2kview_fixedres.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FINGER_AN2KVIEW_FIXEDRES_H__
00012 #define __BE_FINGER_AN2KVIEW_FIXEDRES_H__
00013
00014 #include <be_finger_an2kview.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Finger
00019     {
00036         class AN2KViewFixedResolution : public Finger::AN2KView {
00037         public:
00038
00062             AN2KViewFixedResolution(
00063                 const std::string filename,
00064                 const RecordType typeID,
```

```
00065                 const uint32_t recordNumber);
00066
00087             AN2KViewFixedResolution(
00088                 Memory::uint8Array &buf,
00089                 const RecordType typeID,
00090                 const uint32_t recordNumber);
00091
00092         protected:
00093
00094         private:
00104             void readImageRecord(
00105                 const RecordType typeID);
00106         };
00107     }
00108 }
00109 #endif /* __BE_FINGER_AN2KVIEW_FIXEDRES_H__ */
00110
```

I.26 be_finger_ansi2004view.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FINGER_ANSI2004VIEW_H__
00012 #define __BE_FINGER_ANSI2004VIEW_H__
00013
00014 #include <be_finger_incitsview.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Finger
00019     {
00029         class ANSI2004View : public Finger::INCITSView {
00030         public:
00035             ANSI2004View();
00036
00058             ANSI2004View(
00059                 const std::string &fmrFilename,
00060                     const std::string &firFilename,
00061                 const uint32_t viewNumber);
00062
00085             ANSI2004View(
00086                 const Memory::uint8Array &fmrBuffer,
00087                 const Memory::uint8Array &firBuffer,
00088                 const uint32_t viewNumber);
00089
00090             virtual ~ANSI2004View() = default;
00091
00092         protected:
00093             static const uint32_t BASE_SPEC_VERSION = 0x20323000;
00094             /* ' ' '2' '0' 'nul' */
00095
00096             void readFMRHeader(
00097                 Memory::IndexedBuffer &buf);
00098
00099             /*
00100              * Required implementation of reading core/delta data.
00101              */
00102             void readCoreDeltaData(
00103                 Memory::IndexedBuffer &buf,
00104                 uint32_t dataLength,
00105                 Feature::CorePointSet &cores,
00106                 Feature::DeltaPointSet &deltas);
00107
00108         private:
00109             void init(
00110                 const Memory::uint8Array &fmrBuffer,
00111                 const Memory::uint8Array &firBuffer,
```

```
00112                 const uint32_t viewNumber);
00113         };
00114     }
00115 }
00116 #endif /* __BE_FINGER_ANSI2004VIEW_H__ */
00117
```

I.27 be_finger_ansi2007view.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FINGER_ANSI2007VIEW_H__
00012 #define __BE_FINGER_ANSI2007VIEW_H__
00013
00014 #include <be_finger_incitsview.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Finger
00019     {
00020
00030         class ANSI2007View : public Finger::INCITSView {
00031         public:
00032
00033             ANSI2007View();
00034
00059             ANSI2007View(
00060                 const std::string &fmrFilename,
00061                 const std::string &firFilename,
00062                 const uint32_t viewNumber);
00063
00090             ANSI2007View(
00091                 const Memory::uint8Array &fmrBuffer,
00092                 const Memory::uint8Array &firBuffer,
00093                 const uint32_t viewNumber);
00094
00095         protected:
00096             static const uint32_t BASE_SPEC_VERSION = 0x30333000;
00097             /* '0' '3' '0' 'nul' */
00098
00099             void readFMRHeader(
00100                 Memory::IndexedBuffer &buf);
00101
00102             /*
00103              * Override the inherited method to read the
00104              * Finger View Minutiae Record portion of the
00105              * INCITS/ANSI Finger Minutiae Record.
00106              */
00107             void readFVMR(
00108                 Memory::IndexedBuffer &buf);
00109
00110             /*
00111              * Required implementation of reading core/delta data.
00112              */
00113             void readCoreDeltaData(
00114                 Memory::IndexedBuffer &buf,
00115                 uint32_t dataLength,
00116                 Feature::CorePointSet &cores,
00117                 Feature::DeltaPointSet &deltas);
00118
00119         private:
00120             uint32_t _algorithmID;
00121             void init(
00122                 const Memory::uint8Array &fmrBuffer,
00123                 const Memory::uint8Array &firBuffer,
00124                 const uint32_t viewNumber);
00125
```

```
00126         };
00127     }
00128 }
00129 #endif /* __BE_FINGER_ANSI2007VIEW_H__ */
00130
```

I.28 be_finger_incitsview.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FINGER_INCITSVIEW_H__
00012 #define __BE_FINGER_INCITSVIEW_H__
00013
00014 #include <tuple>
00015
00016 #include <be_view_view.h>
00017 #include <be_feature_incitsminutiae.h>
00018
00019 namespace BiometricEvaluation
00020 {
00021     namespace Finger
00022     {
00023
00036         class INCITSView : public View::View {
00037         public:
00051             static Finger::Position
00052                 convertPosition(int incitsFGP);
00053
00070             static Finger::Impression
00071                 convertImpression(int incitsIMP);
00072
00077             Feature::INCITSMinutiae getMinutiaeData() const;
00078
00085             Finger::Position getPosition() const;
00086
00093             Finger::Impression getImpressionType() const;
00094
00101             uint32_t getQuality() const;
00102
00109             uint16_t getCaptureEquipmentID() const;
00110
00118             inline bool
00119             isAppendixFCompliant()
00120                 const
00121             {
00122                 return (this->_appendixFCompliance);
00123             }
00124
00132             uint16_t
00133             getProductIDOwner()
00134                 const
00135             {
00136                 return (this->_productIDOwner);
00137             }
00138
00146             inline uint16_t
00147             getProductIDType()
00148                 const
00149             {
00150                 return (this->_productIDType);
00151             }
00152
00157             uint32_t
00158             getRecordLength()
00159                 const;
00160
00165             uint8_t
```

```
00166             getNumFingerViews()
00167                 const;
00168
00170             uint8_t
00171             getFMRReservedByte()
00172                 const;
00173
00175             uint32_t
00176             getViewNumber()
00177                 const;
00178
00184             uint16_t
00185             getEDBLength()
00186                 const;
00187
00192             std::vector<uint8_t>
00193             getMinutiaeReservedData()
00194                 const;
00195
00202             void
00203             setMinutiaeData(
00204                 const Feature::INCITSMinutiae &fmd);
00205
00212             void
00213             setMinutiaeReservedData(
00214                 const std::vector<uint8_t> &reservedBits);
00215
00216         protected:
00217
00218             static const uint32_t FMR_BASE_FORMAT_ID = 0x464D5200;
00219             /* 'F' 'M' 'R' 'nul' */
00220
00226             static const uint32_t ANSI2004_STANDARD = 1;
00227             static const uint32_t ISO2005_STANDARD = 2;
00228             static const uint32_t ANSI2007_STANDARD = 3;
00229
00230             INCITSView();
00231
00254             INCITSView(
00255                 const std::string &fmrFilename,
00256                 const std::string &firFilename,
00257                 const uint32_t viewNumber);
00258
00279             INCITSView(
00280                 const Memory::uint8Array &fmrBuffer,
00281                 const Memory::uint8Array &firBuffer,
00282                 const uint32_t viewNumber);
00283
00291             Memory::uint8Array const& getFMRData() const;
00292
00300             Memory::uint8Array const& getFIRData() const;
00301
00308             void setPosition(const Finger::Position &position);
00309
00316             void setImpressionType(
00317                 const Finger::Impression &impression);
00318
00325             void setQuality(uint32_t quality);
00326
00333             void setViewNumber(uint32_t viewNumber);
00334
00341             void setCaptureEquipmentID(uint16_t id);
00342
00351             void setCBEFFProductIDs(uint16_t owner, uint16_t type);
00352
00360             void setAppendixFCompliance(bool flag);
00361
00383             void readFMRHeader(
00384                 Memory::IndexedBuffer &buf,
00385                 const uint32_t formatStandard);
00386
00408             void readFVMR(
00409                 Memory::IndexedBuffer &buf);
00410
00430             virtual
00431             std::tuple<Feature::MinutiaPointSet,
00432             std::vector<uint8_t>>
```

```
00433             readMinutiaeDataPoints(
00434                 Memory::IndexedBuffer &buf,
00435                 uint32_t count);
00436
00447             virtual void readExtendedDataBlock(
00448                 Memory::IndexedBuffer &buf);
00449
00466             virtual Feature::RidgeCountItemSet readRidgeCountData(
00467                 Memory::IndexedBuffer &buf,
00468                 uint32_t dataLength);
00469
00488             virtual void readCoreDeltaData(
00489                 Memory::IndexedBuffer &buf,
00490                 uint32_t dataLength,
00491                 Feature::CorePointSet &cores,
00492                 Feature::DeltaPointSet &deltas) = 0;
00493
00494         private:
00495             Memory::uint8Array _fmr{};
00496             Memory::uint8Array _fir{};
00497             Finger::Position _position{};
00498             Feature::INCITSMinutiae _minutiae{};
00499             std::vector<uint8_t> _fmdReserved{};
00500             Finger::Impression _impression{};
00501             uint32_t _viewNumber{};
00502             uint32_t _quality{};
00503             bool _appendixFCompliance{};
00504             uint16_t _productIDOwner{};
00505             uint16_t _productIDType{};
00506             uint16_t _captureEquipmentID{};
00507             uint32_t _recordLength{};
00508             uint8_t _numFingerViews{};
00509             uint8_t _fmrReservedByte{};
00510             uint16_t _edbLength{};
00511         };
00512     }
00513 }
00514 #endif /* __BE_FINGER_INCITSVIEW_H__ */
00515
```

I.29 be_finger_iso2005view.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FINGER_ISO2005VIEW_H__
00012 #define __BE_FINGER_ISO2005VIEW_H__
00013
00014 #include <be_finger_incitsview.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Finger
00019     {
00029         class ISO2005View : public Finger::INCITSView {
00030         public:
00031             /*
00032              * Construct an empty ISO-2005 view.
00033              */
00034             ISO2005View();
00035
00057             ISO2005View(
00058                 const std::string &fmrFilename,
00059                 const std::string &firFilename,
00060                 const uint32_t viewNumber);
00061
00087             ISO2005View(
00088                 const Memory::uint8Array &fmrBuffer,
```

```
00089                 const Memory::uint8Array &firBuffer,
00090                 const uint32_t viewNumber);
00091
00092         protected:
00093             static const uint32_t BASE_SPEC_VERSION = 0x20323000;
00094             /* ' ' '2' '0' 'nul' */
00095
00096             void readFMRHeader(
00097                 Memory::IndexedBuffer &buf);
00098
00099             /*
00100              * Required implementation of reading core/delta data.
00101              */
00102             void readCoreDeltaData(
00103                 Memory::IndexedBuffer &buf,
00104                 uint32_t dataLength,
00105                 Feature::CorePointSet &cores,
00106                 Feature::DeltaPointSet &deltas);
00107
00108         private:
00109             void init(
00110                 const Memory::uint8Array &fmrBuffer,
00111                 const Memory::uint8Array &firBuffer,
00112                 const uint32_t viewNumber);
00113         };
00114     }
00115 }
00116 #endif /* __BE_FINGER_ISO2005VIEW_H__ */
00117
```

I.30 be_framework.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_FRAMEWORK_H__
00012 #define __BE_FRAMEWORK_H__
00013
00014 #include <string>
00015
00016 namespace BiometricEvaluation
00017 {
00022     namespace Framework
00023     {
00031         unsigned int
00032         getMajorVersion();
00033
00041         unsigned int
00042         getMinorVersion();
00043
00051         std::string
00052         getCompiler();
00053
00062         std::string
00063         getCompileDate();
00064
00073         std::string
00074         getCompileTime();
00075
00083         std::string
00084         getCompilerVersion();
00085     }
00086 }
00087
00088 #endif /* __BE_FRAMEWORK_H__ */
00089
```

I.31 be_framework_api.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to Title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_FRAMEWORK_API_H_
00012 #define BE_FRAMEWORK_API_H_
00013
00014 #include <functional>
00015 #include <memory>
00016
00017 #include <be_error_signal_manager.h>
00018 #include <be_framework_enumeration.h>
00019 #include <be_framework_status.h>
00020 #include <be_time_timer.h>
00021 #include <be_time_watchdog.h>
00022
00023 namespace BiometricEvaluation
00024 {
00025     namespace Framework
00026     {
00028         enum class APICurrentState
00029         {
00031             NeverCalled,
00033             WatchdogExpired,
00035             SignalCaught,
00037             ExceptionCaught,
00039             Running,
00041             Completed
00042         };
00043     }
00044 }
00045
00046 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00047     BiometricEvaluation::Framework::APICurrentState,
00048     BE_Framework_APICurrentState_EnumToStringMap);
00049
00050 namespace BiometricEvaluation
00051 {
00052     namespace Framework
00053     {
00062         template<typename T>
00063         class API
00064         {
00065         public:
00067             class Result
00068             {
00069             public:
00071                 Result();
00072
00074                 std::common_type_t<
00075                     Time::Timer::BE_CLOCK_TYPE::time_point::
00076                     duration,
00077                     Time::Timer::BE_CLOCK_TYPE::time_point::
00078                     duration> elapsedTimePoint;
00087                 T status;
00092                 APICurrentState currentState;
00093
00102                 inline bool
00103                 operator!()
00104                     const
00105                 {
00106                     return (currentState !=
00107                         APICurrentState::Completed);
00108                 }
00109
00117                 inline explicit operator
00118                 bool()
00119                     const
00120                 {
```

```
00121                   return (currentState ==
00122                       APICurrentState::Completed);
00123               }
00124
00135               std::string
00136               getExceptionStr()
00137                   const
00138                   noexcept
00139               {
00140                   try {
00141                       this->rethrowException();
00142                   } catch (const std::exception &e) {
00143                       return (e.what());
00144                   } catch (...) {
00145                       return {};
00146                   }
00147               }
00148
00168               [[noreturn]]
00169               void
00170               rethrowException()
00171                   const
00172               {
00173                   if (this->currentState !=
00174                       APICurrentState::ExceptionCaught)
00175                       throw Error::StrategyError{
00176                           "No exception handled, "
00177                           "current state is " +
00178                           Enumeration::to_string(
00179                           this->currentState)};
00180                   if (!this->exceptionPtr)
00181                       throw Error::StrategyError{
00182                           "Exception was caught, but "
00183                           "not saved"};
00184
00185                   std::rethrow_exception(
00186                       this->exceptionPtr);
00187               }
00188
00196               void
00197               setException(
00198                   std::exception_ptr e)
00199               {
00200                   this->exceptionPtr = e;
00201               }
00202
00207               template<typename Duration>
00208               std::uintmax_t
00209               elapsed()
00210                   const
00211               {
00212                   return (std::chrono::duration_cast<
00213                       Duration>(this->elapsedTimePoint).
00214                       count());
00215               }
00216
00217           private:
00219               std::exception_ptr exceptionPtr{};
00220           };
00221
00223           API();
00224
00259           Result
00260           call(
00261               const std::function<T(void)> &operation,
00262               const std::function<void(const Result&)>
00263               &success = {},
00264               const std::function<void(const Result&)>
00265               &failure = {});
00266
00286           bool
00287           protectionsEnabled()
00288               const
00289           {
00290               return (this->willCatchExceptions() &&
00291                   !this->willRethrowExceptions() &&
00292                   this->getWatchdog()->isEnabled() &&
```

```
00293                   this->getSignalManager()->isEnabled());
00294           }
00295
00315           void
00316           setProtectionsEnabled(
00317               const bool protectionsEnabled)
00318           {
00319               this->setCatchExceptions(protectionsEnabled);
00320               this->setRethrowExceptions(!protectionsEnabled);
00321               this->getWatchdog()->setEnabled(
00322                   protectionsEnabled);
00323               this->getSignalManager()->setEnabled(
00324                   protectionsEnabled);
00325           }
00326
00340           bool
00341           willRethrowExceptions()
00342               const
00343           {
00344               return (this->_rethrowExceptions);
00345           }
00346
00360           void
00361           setRethrowExceptions(
00362               const bool shouldRethrow)
00363           {
00364               this->_rethrowExceptions = shouldRethrow;
00365           }
00366
00376           void
00377           setCatchExceptions(
00378               const bool catchExceptions)
00379           {
00380               this->_catchExceptions = catchExceptions;
00381           }
00382
00392           bool
00393           willCatchExceptions()
00394               const
00395           {
00396               return (this->_catchExceptions);
00397           }
00398
00406           inline std::shared_ptr<BiometricEvaluation::Time::Timer>
00407           getTimer()
00408               noexcept
00409           {
00410               return (_timer);
00411           }
00412
00420           inline std::shared_ptr<
00421               BiometricEvaluation::Time::Watchdog>
00422           getWatchdog()
00423               noexcept
00424           {
00425               return (_watchdog);
00426           }
00427
00435           inline std::shared_ptr<
00436               BiometricEvaluation::Error::SignalManager>
00437           getSignalManager()
00438               noexcept
00439           {
00440               return (_sigmgr);
00441           }
00442
00443       private:
00445           bool _catchExceptions{true};
00447           bool _rethrowExceptions{false};
00449           std::shared_ptr<BiometricEvaluation::Time::Timer>
00450               _timer;
00452           std::shared_ptr<BiometricEvaluation::Time::Watchdog>
00453               _watchdog;
00455           std::shared_ptr<
00456               BiometricEvaluation::Error::SignalManager> _sigmgr;
00457       };
00458   }
```

```
00459 }
00460
00461 template<typename T>
00462 BiometricEvaluation::Framework::API<T>::Result::Result() :
00463     currentState(BiometricEvaluation::Framework::APICurrentState::NeverCalled)
00464 {
00465
00466 }
00467
00468 template<typename T>
00469 BiometricEvaluation::Framework::API<T>::API() :
00470     _catchExceptions{true},
00471     _rethrowExceptions{false},
00472     _timer(new BiometricEvaluation::Time::Timer()),
00473     _watchdog(new BiometricEvaluation::Time::Watchdog(
00474         BiometricEvaluation::Time::Watchdog::REALTIME)),
00475     _sigmgr(new BiometricEvaluation::Error::SignalManager())
00476 {
00477
00478 }
00479
00480 template<typename T>
00481 typename BiometricEvaluation::Framework::API<T>::Result
00482 BiometricEvaluation::Framework::API<T>::call(
00483     const std::function<T(void)> &operation,
00484     const std::function<void(const Framework::API<T>::Result&)> &success,
00485     const std::function<void(const Framework::API<T>::Result&)> &failure)
00486 {
00487     Result ret;
00488
00489     BEGIN_SIGNAL_BLOCK(this->getSignalManager(), SM_BLOCK);
00490     BEGIN_WATCHDOG_BLOCK(this->getWatchdog(), WD_BLOCK);
00491         ret.currentState = APICurrentState::Running;
00492         if (this->willCatchExceptions()) {
00493             this->getTimer()->start();
00494             try {
00495                 ret.status = operation();
00496             } catch (...) {
00497                 this->getTimer()->stop();
00498                 ret.elapsedTimePoint = this->getTimer()->
00499                     elapsedTimePoint();
00500                 ret.currentState =
00501                     APICurrentState::ExceptionCaught;
00502                 ret.setException(std::current_exception());
00503
00504                 if (failure)
00505                     failure(ret);
00506
00507                 if (this->_rethrowExceptions)
00508                     throw;
00509
00510                 return (ret);
00511             }
00512         } else {
00513             this->getTimer()->start();
00514             ret.status = operation();
00515         }
00516         this->getTimer()->stop();
00517     END_WATCHDOG_BLOCK(this->getWatchdog(), WD_BLOCK);
00518     END_SIGNAL_BLOCK(this->getSignalManager(), SM_BLOCK);
00519     if (this->getSignalManager()->sigHandled()) {
00520         this->getTimer()->stop();
00521         ret.elapsedTimePoint = this->getTimer()->elapsedTimePoint();
00522         ret.currentState = APICurrentState::SignalCaught;
00523
00524         if (failure)
00525             failure(ret);
00526     } else if (this->getWatchdog()->expired()) {
00527         this->getTimer()->stop();
00528         ret.elapsedTimePoint = this->getTimer()->elapsedTimePoint();
00529         ret.currentState = APICurrentState::WatchdogExpired;
00530
00531         if (failure)
00532             failure(ret);
00533     } else {
00534         ret.currentState = APICurrentState::Completed;
00535         ret.elapsedTimePoint = this->getTimer()->elapsedTimePoint();
```

```
00536
00537         if (success)
00538             success(ret);
00539     }
00540
00541     return (ret);
00542 }
00543
00544 #endif /* BE_FRAMEWORK_API_H_ */
```

I.32 be_framework_enumeration.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_FRAMEWORK_ENUMERATION_H_
00012 #define BE_FRAMEWORK_ENUMERATION_H_
00013
00014 #include <map>
00015 #include <string>
00016 #include <ostream>
00017 #include <type_traits>
00018
00019 #include <be_error_exception.h>
00020
00021 /*
00022  * These empty namespaces are here so that you can issue
00023  *  using namespace BiometricEvaluation::Framework::Enumeration;
00024  * after including this file.
00025  */
00026 namespace BiometricEvaluation
00027 {
00028     namespace Framework
00029     {
00030         namespace Enumeration
00031         {
00032
00033         }
00034     }
00035 }
00036
00049 #define BE_FRAMEWORK_ENUMERATION_DECLARATIONS(BE_ENUMERATED_TYPE_, \
00050     BE_ENUMERATED_TYPE_ENUM_TO_STRING_MAP_) \
00051 extern const \
00052 std::map<BE_ENUMERATED_TYPE_, std::string> \
00053 BE_ENUMERATED_TYPE_ENUM_TO_STRING_MAP_;\
00054 \
00055 namespace BiometricEvaluation \
00056 { \
00057     namespace Framework \
00058     { \
00059         namespace Enumeration \
00060         { \
00061
00076  \
00077             bool \
00078             operator==( \
00079                 const std::string &strVal, \
00080                 const BE_ENUMERATED_TYPE_ &enumVal); \
00081 \
00082
00097  \
00098             bool \
00099             operator==( \
00100                 const BE_ENUMERATED_TYPE_ &enumVal, \
00101                 const std::string &strVal); \
00102 \
00103
```

```
00118  \
00119             bool \
00120             operator!=( \
00121                 const std::string &strVal, \
00122                 const BE_ENUMERATED_TYPE_ &enumVal); \
00123 \
00124
00139  \
00140             bool \
00141             operator!=( \
00142                 const BE_ENUMERATED_TYPE_ &enumVal, \
00143                 const std::string &strVal); \
00144 \
00145
00158  \
00159             std::ostream& \
00160             operator<<( \
00161                 std::ostream &stream, \
00162                 const BE_ENUMERATED_TYPE_ &enumVal); \
00163 \
00164
00178  \
00179             std::string \
00180             operator+( \
00181                 const std::string &strVal, \
00182                 const BE_ENUMERATED_TYPE_ &enumVal); \
00183 \
00184
00198  \
00199             std::string \
00200             operator+( \
00201                 const BE_ENUMERATED_TYPE_ &enumVal, \
00202                 const std::string &strVal); \
00203 \
00204
00215  \
00216             std::underlying_type<BE_ENUMERATED_TYPE_>::type \
00217             to_int_type( \
00218                 const BE_ENUMERATED_TYPE_ &enumVal) \
00219                 noexcept; \
00220 \
00221
00237  \
00238             std::string \
00239             to_string( \
00240                 const BE_ENUMERATED_TYPE_ &enumVal); \
00241 \
00242
00257  \
00258             template<typename T> \
00259             T \
00260             to_enum( \
00261                 const typename \
00262                 std::underlying_type<T>::type &iVal); \
00263 \
00264
00288  \
00289             template<typename T> \
00290             T \
00291             to_enum( \
00292                 const std::string &strVal); \
00293         } \
00294     } \
00295 }\
00296 /* This is here to require a semicolon after macro instantation */\
00297 static_assert(true, "")
00298
00299
00312 #define BE_FRAMEWORK_ENUMERATION_DEFINITIONS(BE_ENUMERATED_TYPE_, \
00313     BE_ENUMERATED_TYPE_ENUM_TO_STRING_MAP_) \
00314 /* \
00315  * Template specializations for to_enum() must come before they are used, \
00316  * or functions that rely on them, like operator==, will implicitly \
00317  * instantiate them, disallowing the later specialization. \
00318  *
00319  * Additional, a long-standing g++ bug originating from a defect in the \
00320  * standard requires that the template specializations be enclosed in the same \
```

```
00321  * namespace: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=56480 \
00322  */ \
00323 namespace BiometricEvaluation{ \
00324     namespace Framework { \
00325         namespace Enumeration{ \
00326             template<> \
00327             BE_ENUMERATED_TYPE_ \
00328             to_enum( \
00329                 const typename std::underlying_type<\
00330                     BE_ENUMERATED_TYPE_>::type &iVal) \
00331             { \
00332                 for (const auto &i : \
00333                     BE_ENUMERATED_TYPE_ENUM_TO_STRING_MAP_) \
00334                     if (static_cast<std::underlying_type<\
00335                         BE_ENUMERATED_TYPE_>::type>(\
00336                         i.first) == iVal) \
00337                         return (i.first); \
00338             \
00339                 throw BiometricEvaluation::Error::\
00340                     ObjectDoesNotExist(std::to_string(iVal)); \
00341             } \
00342 \
00343             template<> \
00344             BE_ENUMERATED_TYPE_ \
00345             to_enum ( \
00346                 const std::string &strVal) \
00347             { \
00348                 for (const auto &i : \
00349                     BE_ENUMERATED_TYPE_ENUM_TO_STRING_MAP_) \
00350                     if (i.second == strVal) \
00351                         return (i.first); \
00352             \
00353                 throw BiometricEvaluation::Error::\
00354                     ObjectDoesNotExist(strVal); \
00355             } \
00356         } \
00357     } \
00358 } \
00359 \
00360 bool \
00361 BiometricEvaluation::Framework::Enumeration::operator==( \
00362     const std::string &strVal, \
00363     const BE_ENUMERATED_TYPE_ &enumVal) \
00364 { \
00365     return (to_enum<BE_ENUMERATED_TYPE_>(strVal) == enumVal); \
00366 } \
00367 \
00368 bool \
00369 BiometricEvaluation::Framework::Enumeration::operator==( \
00370     const BE_ENUMERATED_TYPE_ &enumVal, \
00371     const std::string &strVal) \
00372 { \
00373     return (strVal == enumVal); \
00374 } \
00375 \
00376 bool \
00377 BiometricEvaluation::Framework::Enumeration::operator!=( \
00378     const std::string &strVal, \
00379     const BE_ENUMERATED_TYPE_ &enumVal) \
00380 { \
00381     return (!(strVal == enumVal)); \
00382 } \
00383 \
00384 bool \
00385 BiometricEvaluation::Framework::Enumeration::operator!=( \
00386     const BE_ENUMERATED_TYPE_ &enumVal, \
00387     const std::string &strVal) \
00388 { \
00389     return (!(strVal == enumVal)); \
00390 } \
00391 \
00392 std::ostream& \
00393 BiometricEvaluation::Framework::Enumeration::operator<<( \
00394     std::ostream &stream, \
00395     const BE_ENUMERATED_TYPE_ &enumVal) \
00396 { \
00397     return (stream << to_string(enumVal)); \
```

```
00398 } \
00399 \
00400 std::string \
00401 BiometricEvaluation::Framework::Enumeration::operator+( \
00402     const std::string &strVal, \
00403     const BE_ENUMERATED_TYPE_ &enumVal) \
00404 { \
00405     return (strVal + to_string(enumVal)); \
00406 } \
00407 \
00408 std::string \
00409 BiometricEvaluation::Framework::Enumeration::operator+( \
00410     const BE_ENUMERATED_TYPE_ &enumVal, \
00411     const std::string &strVal) \
00412 { \
00413     return (to_string(enumVal) + strVal); \
00414 } \
00415 \
00416 std::string \
00417 BiometricEvaluation::Framework::Enumeration::to_string( \
00418     const BE_ENUMERATED_TYPE_ &enumVal) \
00419 { \
00420     return (BE_ENUMERATED_TYPE_ENUM_TO_STRING_MAP_.at(enumVal)); \
00421 } \
00422 \
00423 std::underlying_type<BE_ENUMERATED_TYPE_>::type \
00424 BiometricEvaluation::Framework::Enumeration::to_int_type( \
00425     const BE_ENUMERATED_TYPE_ &enumVal) \
00426     noexcept \
00427 { \
00428     return (static_cast<typename std::underlying_type< \
00429         BE_ENUMERATED_TYPE_>::type>(enumVal)); \
00430 }\
00431 /* This is here to require a semicolon after macro instantation */\
00432 static_assert(true, "")
00433
00434 #endif /* BE_FRAMEWORK_ENUMERATION_H_ */
```

I.33 be_framework_status.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_FRAMEWORK_STATUS_H_
00012 #define BE_FRAMEWORK_STATUS_H_
00013
00014 #include <ostream>
00015 #include <string>
00016
00017 #include <be_framework_enumeration.h>
00018
00019 namespace BiometricEvaluation
00020 {
00021     namespace Framework
00022     {
00024         class Status
00025         {
00026         public:
00028             enum class Type
00029             {
00034                 Debug,
00039                 Warning,
00041                 Error
00042             };
00043
00053             Status(
00054                 Type type,
00055                 const std::string &message,
```

```
00056                 const std::string &identifier = "");
00057
00065             inline Type
00066             getType()
00067                 const
00068                 noexcept
00069             {
00070                 return (this->_type);
00071             }
00072
00083             inline std::string
00084             getMessage()
00085                 const
00086                 noexcept
00087             {
00088                 return (this->_message);
00089             }
00090
00104             inline std::string
00105             getIdentifier()
00106                 const
00107                 noexcept
00108             {
00109                 return (this->_identifier);
00110             }
00111
00112         private:
00114             Type _type{Type::Debug};
00116             std::string _message{};
00118             std::string _identifier{};
00119         };
00120
00131         std::string
00132         to_string(
00133             const Status &status);
00134
00147         std::ostream&
00148         operator<<(
00149             std::ostream &s,
00150             const Status &status);
00151     }
00152 }
00153
00154 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00155     BiometricEvaluation::Framework::Status::Type,
00156     BE_Framework_Status_Type_EnumToStringMap);
00157
00158 #endif /* BE_FRAMEWORK_STATUS_H_ */
```

I.34 be_image.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IMAGE_H__
00012 #define __BE_IMAGE_H__
00013
00014 #include <cstdint>
00015 #include <iostream>
00016 #include <vector>
00017
00018 #include <be_framework_enumeration.h>
00019 #include <be_memory_autoarray.h>
00020
00021 namespace BiometricEvaluation
00022 {
00031     namespace Image
00032     {
```

```
00034         enum class CompressionAlgorithm
00035         {
00036             None        = 0,
00037             Facsimile   = 1,
00038             WSQ20       = 2,
00039             JPEGB       = 3,
00040             JPEGL      = 4,
00041             JP2     = 5,
00042             JP2L        = 6,
00043             PNG      = 7,
00044             NetPBM        = 8,
00045             BMP      = 9,
00046             TIFF        = 10
00047         };
00048
00050         enum class PixelFormat
00051         {
00053             MonoWhite  = 0,
00055             MonoBlack  = 1,
00057             Gray8      = 2,
00059             RGB24      = 3
00060         };
00061
00067         struct Coordinate {
00081             Coordinate(
00082                 const uint32_t x = 0,
00083                 const uint32_t y = 0,
00084                 const float xDistance = 0,
00085                 const float yDistance = 0);
00086
00088             uint32_t x;
00090             uint32_t y;
00092             float xDistance;
00094             float yDistance;
00095         };
00096         using Coordinate = struct Coordinate;
00097
00108         std::string
00109         to_string(
00110             const Coordinate &c);
00111         std::ostream& operator<< (std::ostream&, const Coordinate&);
00112         using CoordinateSet = std::vector<Image::Coordinate>;
00113         bool
00114         operator==(
00115             const Coordinate &lhs,
00116             const Coordinate &rhs);
00117         bool
00118         operator!=(
00119             const Coordinate &lhs,
00120             const Coordinate &rhs);
00121
00132         std::string
00133         to_string(
00134             const CoordinateSet &coordinates);
00149         std::ostream&
00150         operator<<(
00151             std::ostream &stream,
00152             const CoordinateSet &coordinates);
00153
00159         struct Size {
00169             Size(
00170                 const uint32_t xSize = 0,
00171                 const uint32_t ySize = 0);
00172
00174             uint32_t xSize;
00176             uint32_t ySize;
00177         };
00178         using Size = struct Size;
00189         std::string
00190         to_string(
00191             const Size &s);
00192         std::ostream& operator<< (std::ostream&, const Size&);
00193         bool
00194         operator==(
00195             const Size &lhs,
00196             const Size &rhs);
00197         bool
```

```
00198         operator!=(
00199             const Size &lhs,
00200             const Size &rhs);
00201
00206         struct Resolution {
00212             enum class Units {
00214                 NA    = 0,
00216                 PPI  = 1,
00218                 PPMM    = 2,
00220                 PPCM    = 3
00221             };
00222
00234             Resolution(
00235                 const double xRes = 0.0,
00236                 const double yRes = 0.0,
00237                 const Units units = Units::PPI);
00238
00240             double xRes;
00242             double yRes;
00244             Units units;
00245
00260             Resolution
00261             toUnits(
00262                 const Units &units)
00263                 const;
00264         };
00265         using Resolution = struct Resolution;
00266
00268         const double CentimetersPerInch = 2.54;
00270         const double MillimetersPerInch = CentimetersPerInch * 10;
00271
00282         std::string
00283         to_string(
00284             const Resolution &r);
00285         std::ostream& operator<< (std::ostream&, const Resolution&);
00286         bool
00287         operator==(
00288             const Resolution &lhs,
00289             const Resolution &rhs);
00290         bool
00291         operator!=(
00292             const Resolution &lhs,
00293             const Resolution &rhs);
00294
00307         float
00308         distance(
00309             const Coordinate &p1,
00310             const Coordinate &p2);
00311
00337         BiometricEvaluation::Memory::uint8Array
00338         removeComponents(
00339             const BiometricEvaluation::Memory::uint8Array &rawData,
00340             const uint8_t bitDepth,
00341             const std::vector<bool> &components);
00342
00348         struct ROI {
00352             ROI();
00353
00365             ROI(
00366                 const Size size,
00367                 const uint32_t horzOffset,
00368                 const uint32_t vertOffset,
00369                 const CoordinateSet &path);
00370
00371             Size size;
00372             uint32_t horzOffset;
00373             uint32_t vertOffset;
00374             CoordinateSet path;
00375         };
00376         using ROI = struct ROI;
00377
00388         std::string
00389         to_string(
00390             const ROI &r);
00391         std::ostream& operator<< (std::ostream&, const ROI&);
00392         bool
00393         operator==(
```

```
00394             const ROI &lhs,
00395             const ROI &rhs);
00396         bool
00397         operator!=(
00398             const ROI &lhs,
00399             const ROI &rhs);
00400     }
00401 }
00402
00403 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00404     BiometricEvaluation::Image::CompressionAlgorithm,
00405     BE_Image_CompressionAlgorithm_EnumToStringMap);
00406
00407 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00408     BiometricEvaluation::Image::PixelFormat,
00409     BE_Image_PixelFormat_EnumToStringMap);
00410
00411 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00412     BiometricEvaluation::Image::Resolution::Units,
00413     BE_Image_Resolution_Units_EnumToStringMap);
00414
00415 #endif /* __BE_IMAGE_H__ */
```

I.35 be_image_bmp.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IMAGE_BMP__
00012 #define __BE_IMAGE_BMP__
00013
00014 #include <be_image_image.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Image
00019     {
00029         class BMP : public Image
00030         {
00031         public:
00033             struct ColorTableEntry
00034             {
00036                 uint8_t     red;
00038                 uint8_t     green;
00040                 uint8_t     blue;
00042                 uint8_t     reserved;
00043             };
00044             using ColorTableEntry = struct ColorTableEntry;
00045             using ColorTable = std::vector<ColorTableEntry>;
00046
00047             BMP(
00048                 const uint8_t *data,
00049                 const uint64_t size,
00050                 const std::string &identifier = "",
00051                 const statusCallback_t &statusCallback =
00052                     Image::defaultStatusCallback);
00053
00054             BMP(
00055                 const Memory::uint8Array &data,
00056                 const std::string &identifier = "",
00057                 const statusCallback_t &statusCallback =
00058                     Image::defaultStatusCallback);
00059
00060             ~BMP() = default;
00061
00062             Memory::AutoArray<uint8_t>
00063             getRawData()
00064                 const;
```

```
00065
00066             Memory::AutoArray<uint8_t>
00067             getRawGrayscaleData(
00068                 uint8_t depth)
00069                 const;
00070
00083             static bool
00084             isBMP(
00085                 const uint8_t *data,
00086                 uint64_t size);
00087         protected:
00088
00089         private:
00091             typedef struct
00092             {
00094                 uint16_t magic;
00096                 uint32_t size;
00098                 uint16_t reserved1;
00100                uint16_t reserved2;
00102                uint32_t startingAddress;
00103            } BMPHeader;
00104
00106            typedef struct
00107            {
00109                uint32_t headerSize;
00111                int32_t width;
00113                int32_t height;
00115                uint16_t colorPanes;
00117                uint16_t bitsPerPixel;
00119                uint32_t compressionMethod;
00121                uint32_t bitmapSize;
00123                uint32_t xResolution;
00125                uint32_t yResolution;
00127                uint32_t numberOfColors;
00129                uint32_t numberOfImportantColors;
00130            } BITMAPINFOHEADER;
00131
00147            static void
00148            getBMPHeader(
00149                const uint8_t * const buf,
00150                uint64_t bufsz,
00151                BMPHeader *header);
00152
00172            static void
00173            getDIBHeader(
00174                const uint8_t * const buf,
00175                uint64_t bufsz,
00176                BITMAPINFOHEADER *header);
00177
00193            void
00194            getColorTable(
00195                const uint8_t *buf,
00196                uint64_t bufsz,
00197                int count,
00198                ColorTable &colorTable);
00199
00219            void
00220            rle8Decoder(
00221                const uint8_t *input,
00222                uint64_t inputSize,
00223                Memory::uint8Array &output,
00224                BMPHeader *bmpHeader,
00225                BITMAPINFOHEADER *dibHeader) const;
00226
00227            ColorTable _colorTable{};
00228        };
00229
00230        #ifndef BI_RGB
00232        static const uint8_t BI_RGB = 0;
00233        #endif /* BI_RGB */
00234        #ifndef BI_RLE8
00236        static const uint8_t BI_RLE8 = 1;
00237        #endif /* BI_RLE8 */
00238
00239    }
00240 }
00241 #endif
```

00242

I.36 be_image_image.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IMAGE_IMAGE_H__
00012 #define __BE_IMAGE_IMAGE_H__
00013
00014 #include <cstdint>
00015 #include <functional>
00016 #include <stdexcept>
00017 #include <memory>
00018
00019 #include <be_framework_status.h>
00020 #include <be_io.h>
00021 #include <be_image.h>
00022 #include <be_memory_autoarray.h>
00023
00024 namespace BiometricEvaluation
00025 {
00030     namespace Image
00031     {
00032         /* Forward declaration */
00033         class Raw;
00034
00050         class Image {
00051         public:
00052             using statusCallback_t = std::function<void(
00053                 const Framework::Status)>;
00054
00086             Image(
00087                 const uint8_t *data,
00088                 const uint64_t size,
00089                 const Size dimensions,
00090                 const uint32_t colorDepth,
00091                 const uint16_t bitDepth,
00092                 const Resolution resolution,
00093                 const CompressionAlgorithm compression,
00094                 const bool hasAlphaChannel,
00095                 const std::string &identifier = "",
00096                 const statusCallback_t &statusCallback =
00097                     Image::defaultStatusCallback);
00098
00120             Image(
00121                 const uint8_t *data,
00122                 const uint64_t size,
00123                 const CompressionAlgorithm compression,
00124                     const std::string &identifier = "",
00125                 const statusCallback_t &statusCallback =
00126                     Image::defaultStatusCallback);
00127
00136             CompressionAlgorithm
00137             getCompressionAlgorithm()
00138                 const;
00139
00147             Resolution
00148             getResolution()
00149                 const;
00150
00159             Memory::uint8Array
00160             getData()
00161                 const;
00162
00180             virtual Memory::uint8Array
00181             getRawData()
00182                 const = 0;
```

```
00183
00210           virtual Memory::uint8Array
00211           getRawData(
00212               const bool removeAlphaChannelIfPresent)
00213               const;
00214
00249           virtual Memory::uint8Array
00250           getRawGrayscaleData(
00251               uint8_t depth)
00252               const = 0;
00253
00262           Size
00263           getDimensions()
00264               const;
00265
00273           uint32_t
00274           getColorDepth()
00275               const;
00276
00284           uint16_t
00285           getBitDepth()
00286               const;
00287
00295           bool
00296           hasAlphaChannel()
00297               const
00298           {
00299               return (this->_hasAlphaChannel);
00300           }
00301
00309           statusCallback_t
00310           getStatusCallback()
00311               const;
00312
00320           std::string
00321           getIdentifier()
00322               const;
00323
00324           virtual ~Image();
00325
00326           /*
00327            * Buffer type conversions.
00328            */
00329
00345           static uint64_t
00346           valueInColorspace(
00347               uint64_t color,
00348               uint64_t maxColorValue,
00349               uint8_t depth);
00350
00351           /*
00352            * Static functions.
00353            */
00354
00378           static std::shared_ptr<Image>
00379           openImage(
00380               const uint8_t *data,
00381               const uint64_t size,
00382               const std::string &identifier = "",
00383               const statusCallback_t &statusCallback =
00384                   Image::defaultStatusCallback);
00385
00407           static std::shared_ptr<Image>
00408           openImage(
00409               const Memory::uint8Array &data,
00410               const std::string &identifier = "",
00411               const statusCallback_t &statusCallback =
00412                   Image::defaultStatusCallback);
00413
00435           static std::shared_ptr<Image>
00436           openImage(
00437               const std::string &path,
00438               const statusCallback_t &statusCallback =
00439                   Image::defaultStatusCallback);
00440
00459           static CompressionAlgorithm
00460           getCompressionAlgorithm(
```

```
00461                 const uint8_t *data,
00462                 const uint64_t size);
00463
00480             static CompressionAlgorithm
00481             getCompressionAlgorithm(
00482                 const Memory::uint8Array &data);
00483
00505             static CompressionAlgorithm
00506             getCompressionAlgorithm(
00507                 const std::string &path);
00508
00523             static BiometricEvaluation::Image::Raw
00524             getRawImage(
00525                 const std::shared_ptr<BiometricEvaluation::Image::
00526                 Image> &image);
00527
00544             static void
00545             defaultStatusCallback(
00546                 const Framework::Status &status);
00547
00548         protected:
00556             void
00557             setResolution(
00558                 const Resolution resolution);
00559
00567             void
00568             setDimensions(
00569                 const Size dimensions);
00570
00578             void
00579             setColorDepth(
00580                 const uint32_t colorDepth);
00581
00590             void
00591             setBitDepth(
00592                 const uint16_t bitDepth);
00593
00595             const uint8_t *
00596             getDataPointer()
00597                 const;
00598
00600             uint64_t
00601             getDataSize()
00602                 const;
00603
00611             void
00612             setHasAlphaChannel(
00613                 const bool hasAlphaChannel)
00614             {
00615                 this->_hasAlphaChannel = hasAlphaChannel;
00616             }
00617
00618         private:
00620             Size _dimensions;
00621
00623             uint32_t _colorDepth;
00624
00626             bool _hasAlphaChannel;
00627
00629             uint16_t _bitDepth;
00630
00632             Resolution _resolution;
00633
00635             Memory::AutoArray<uint8_t> _data;
00636
00638             CompressionAlgorithm _compressionAlgorithm;
00639
00641             const std::string _identifier{};
00642
00644             statusCallback_t _statusCallback{
00645                 Image::defaultStatusCallback};
00646         };
00647     }
00648 }
00649
00650 #endif /* __BE_IMAGE_IMAGE_H__ */
```

I.37 be_image_jpeg.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IMAGE_JPEG__
00012 #define __BE_IMAGE_JPEG__
00013
00014 #include <cstdio>
00015
00016 #include <be_image_image.h>
00017
00018 /* jpeglib.h forward-declares */
00019 extern "C" {
00020 /* libjpeg 9b no longer needs "boolean" defined */
00021 #if JPEG_LIB_VERSION <= 90
00022     #if JPEG_LIB_VERSION == 90
00023         #if JPEG_LIB_VERSION_MINOR < 2
00024             #ifdef _WIN32
00025                 typedef unsigned char boolean;
00026             #else
00027                 typedef int boolean;
00028             #endif /* _WIN32 */
00029         #endif /* JPEG_LIB_VERSION_MINOR < 2 */
00030     #else /* JPEG_LIB_VERSION == 90 */
00031         #ifndef HAVE_BOOLEAN
00032             #ifdef _WIN32
00033                 typedef unsigned char boolean;
00034             #else
00035                 typedef int boolean;
00036             #endif /* _WIN32 */
00037         #endif /* HAVE_BOOLEAN */
00038     #endif /* JPEG_LIB_VERSION == 90 */
00039 #endif /* JPEG_LIB_VERSION */
00040
00041     struct jpeg_decompress_struct;
00042     struct jpeg_common_struct;
00043     typedef struct jpeg_common_struct *j_common_ptr;
00044     typedef struct jpeg_decompress_struct *j_decompress_ptr;
00045 }
00046
00047 namespace BiometricEvaluation
00048 {
00049     namespace Image
00050     {
00055         class JPEG : public Image
00056         {
00057         public:
00058             JPEG(
00059                 const uint8_t *data,
00060                 const uint64_t size,
00061                 const std::string &identifier = "",
00062                 const statusCallback_t &statusCallback =
00063                     Image::defaultStatusCallback);
00064
00065             JPEG(
00066                 const Memory::uint8Array &data,
00067                 const std::string &identifier = "",
00068                 const statusCallback_t &statusCallback =
00069                     Image::defaultStatusCallback);
00070
00071             ~JPEG() = default;
00072
00073             Memory::uint8Array
00074             getRawGrayscaleData(
00075                 uint8_t depth) const;
00076
00077             Memory::uint8Array
00078             getRawData()
00079                 const;
```

```
00080
00093             static bool
00094             isJPEG(
00095                 const uint8_t *data,
00096                 uint64_t size);
00097
00098             static int
00099             getc_skip_marker_segment(
00100                 const unsigned short marker,
00101                 unsigned char **cbufptr,
00102                 unsigned char *ebufptr);
00103
00104         protected:
00105
00106         private:
00116             static void
00117             callStatusCallback(
00118                 const j_common_ptr cinfo,
00119                 const Framework::Status::Type statusType);
00120
00131             static void
00132             error_exit(
00133                 j_common_ptr cinfo);
00134
00143             static void
00144             output_message(
00145                 j_common_ptr cinfo);
00146
00156             static void
00157             emit_message(
00158                 j_common_ptr cinfo,
00159                 int msg_level);
00160
00161             /*
00162              * libjpeg 8.0 has code for handling a JPEG image
00163              * in a buffer location, so don't compile ours.
00164              */
00165 #if JPEG_LIB_VERSION < 80
00166             /*
00167              * JPEG memory source manager based on
00168              * http://stackoverflow.com/questions/5280756/
00169              * libjpeg-ver-6b-jpeg-stdio-src-vs-jpeg-mem-src
00170              */
00171             static void
00172             jpeg_mem_src(
00173                 j_decompress_ptr cinfo,
00174                 uint8_t *buffer,
00175                 long size);
00176
00177             static void
00178             init_source_mem(
00179                 j_decompress_ptr cinfo);
00180
00181             static boolean
00182             fill_input_buffer_mem(
00183                 j_decompress_ptr cinfo);
00184
00185             static void
00186             skip_input_data_mem(
00187                 j_decompress_ptr cinfo,
00188                 long num_bytes);
00189
00190             static void
00191             term_source_mem(
00192                 j_decompress_ptr cinfo);
00193 #endif /* JPEG_LIB_VERSION */
00194         };
00195     }
00196 }
00197
00198 #endif /* __BE_IMAGE_JPEG__ */
00199
```

I.38 be_image_jpeg2000.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IMAGE_JPEG2000__
00012 #define __BE_IMAGE_JPEG2000__
00013
00014 #include <be_image_image.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Image
00019     {
00024         class JPEG2000 : public Image
00025         {
00026         public:
00046             JPEG2000(
00047                 const uint8_t *data,
00048                 const uint64_t size,
00049                 const std::string &identifier = "",
00050                 const statusCallback_t &statusCallback =
00051                     Image::defaultStatusCallback,
00052                 const int8_t codecFormat = 2);
00053
00054             JPEG2000(
00055                 const Memory::uint8Array &data,
00056                 const std::string &identifier = "",
00057                 const statusCallback_t &statusCallback =
00058                     Image::defaultStatusCallback);
00059
00060             ~JPEG2000() = default;
00061
00062             Memory::uint8Array
00063             getRawData()
00064                 const;
00065
00066             Memory::uint8Array
00067             getRawGrayscaleData(
00068                 uint8_t depth) const;
00069
00082             static bool
00083             isJPEG2000(
00084                 const uint8_t *data,
00085                 uint64_t size);
00086
00087         private:
00089             const int8_t _codecFormat;
00090
00102             bool
00103             checkForAlphaInCDEF();
00104
00115             static void
00116             openjpeg_error(
00117                 const char *msg,
00118                 void *client_data);
00119
00130             static void
00131             openjpeg_warning(
00132                 const char *msg,
00133                 void *client_data);
00134
00145             static void
00146             openjpeg_info(
00147                 const char *msg,
00148                 void *client_data);
00149
00169             uint64_t
00170             static find_marker_offset(
00171                 const uint8_t *marker,
```

```
00172                 uint64_t marker_size,
00173                 const uint8_t *buffer,
00174                 uint64_t buffer_size);
00175
00199             Memory::AutoArray<uint8_t>
00200             static find_marker(
00201                 const uint8_t *marker,
00202                 uint64_t marker_size,
00203                 const uint8_t *buffer,
00204                 uint64_t buffer_size,
00205                 uint64_t value_size);
00206
00225             Resolution
00226             parse_res(
00227                 const Memory::AutoArray<uint8_t> &res);
00228
00229             /*
00230              * libopenjp2 stream callbacks.
00231              *
00232              * Note that libopenjp2 types have been converted to
00233              * void* below so that openjpeg.h does not need to
00234              * be in the include path of applications that use this
00235              * file.
00236              */
00237
00239             void*
00240             getDecompressionStream()
00241                 const;
00242
00244             void*
00245             getDecompressionCodec()
00246                 const;
00247
00248             /*
00249              * libopenjp2 callbacks.
00250              *
00251              * Note that libopenjp2 types have been converted
00252              * to standard types below so that openjpeg.h
00253              * does not need to be in the include path of
00254              * applications that use this file.
00255              */
00256
00264             static void
00265             libopenjp2Free(
00266                 void *p_user_data);
00267
00282             static size_t
00283             libopenjp2Read(
00284                 void *p_buffer,
00285                 size_t p_nb_bytes,
00286                 void *p_user_data);
00287
00300             static int64_t
00301             libopenjp2Skip(
00302                 int64_t p_nb_bytes,
00303                 void *p_user_data);
00304
00317             static int
00318             libopenjp2Seek(
00319                 int64_t p_nb_bytes,
00320                 void *p_user_data);
00321         };
00322     }
00323 }
00324
00325 #endif /* __BE_IMAGE_JPEG2000__ */
00326
```

I.39 be_image_jpegl.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
```

```
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IMAGE_JPEGL__
00012 #define __BE_IMAGE_JPEGL__
00013
00014 #include <be_image_image.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Image
00019     {
00024         class JPEGL : public Image
00025         {
00026         public:
00027             JPEGL(
00028                 const uint8_t *data,
00029                 const uint64_t size,
00030                 const std::string &identifier = "",
00031                 const statusCallback_t &statusCallback =
00032                     Image::defaultStatusCallback);
00033
00034             JPEGL(
00035                 const Memory::uint8Array &data,
00036                 const std::string &identifier = "",
00037                 const statusCallback_t &statusCallback =
00038                     Image::defaultStatusCallback);
00039
00040             ~JPEGL() = default;
00041
00042             Memory::uint8Array
00043             getRawGrayscaleData(
00044                 uint8_t depth) const;
00045
00046             Memory::uint8Array
00047             getRawData()
00048                 const;
00049
00062             static bool
00063             isJPEGL(
00064                 const uint8_t *data,
00065                 uint64_t size);
00066
00067         protected:
00068
00069         private:
00070
00071         };
00072     }
00073 }
00074
00075 #endif /* __BE_IMAGE_JPEGL__ */
00076
```

I.40 be_image_netpbm.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IMAGE_NETPBM__
00012 #define __BE_IMAGE_NETPBM__
00013
00014 #include <stdexcept>
00015
00016 #include <be_framework_enumeration.h>
00017 #include <be_image_image.h>
```

```
00018
00019 namespace BiometricEvaluation
00020 {
00021     namespace Image
00022     {
00032         class NetPBM : public Image
00033         {
00034         public:
00035             enum class Kind {
00036                 ASCIIPortableBitmap = 1,    /* P1 */
00037                 ASCIIPortableGraymap = 2,   /* P2 */
00038                 ASCIIPortablePixmap = 3,    /* P3 */
00039                 BinaryPortableBitmap = 4,   /* P4 */
00040                 BinaryPortableGraymap = 5,  /* P5 */
00041                 BinaryPortablePixmap = 6    /* P6 */
00042             };
00043
00044             NetPBM(
00045                 const uint8_t *data,
00046                 const uint64_t size,
00047                 const std::string &identifier = "",
00048                 const statusCallback_t &statusCallback =
00049                     Image::defaultStatusCallback);
00050
00051             NetPBM(
00052                 const Memory::uint8Array &data,
00053                 const std::string &identifier = "",
00054                 const statusCallback_t &statusCallback =
00055                     Image::defaultStatusCallback);
00056
00057             ~NetPBM() = default;
00058
00078             Memory::uint8Array
00079             getRawData()
00080                 const;
00081
00082             Memory::uint8Array
00083             getRawGrayscaleData(
00084                 uint8_t depth) const;
00085
00098             static bool
00099             isNetPBM(
00100                 const uint8_t *data,
00101                 uint64_t size);
00102
00103             /*
00104              * Utility methods for parsing buffers.
00105              */
00106
00124             static void
00125             skipLine(
00126                 const uint8_t *data,
00127                 size_t dataSize,
00128                 size_t &offset);
00129
00146             static void
00147             skipComment(
00148                 const uint8_t *data,
00149                 size_t dataSize,
00150                 size_t &offset);
00151
00172             static std::string
00173             getNextValue(
00174                 const uint8_t *data,
00175                 size_t dataSize,
00176                 size_t &offset,
00177                 size_t sizeOfValue = 0);
00178
00179             /*
00180              * Buffer type conversions.
00181              */
00182
00203             static Memory::uint8Array
00204             ASCIIBitmapTo8Bit(
00205                 const uint8_t *bitmap,
00206                 uint64_t bitmapSize,
00207                 uint32_t width,
```

```
00208                 uint32_t height);
00209
00239             static Memory::uint8Array
00240             ASCIIPixmapToBinaryPixmap(
00241                 const uint8_t *ASCIIBuf,
00242                 uint64_t ASCIIBufSize,
00243                 uint32_t width,
00244                 uint32_t height,
00245                 uint8_t depth,
00246                 uint32_t maxColor);
00247
00268             static Memory::uint8Array
00269             BinaryBitmapTo8Bit(
00270                 const uint8_t *bitmap,
00271                 uint64_t bitmapSize,
00272                 uint32_t width,
00273                 uint32_t height);
00274
00275         private:
00286             void
00287             parseHeader();
00288
00290             uint32_t _maxColorValue;
00292             uint64_t _headerLength;
00294             Kind _kind;
00295         };
00296     }
00297 }
00298
00299 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00300     BiometricEvaluation::Image::NetPBM::Kind,
00301     BE_Image_NetPBM_Kind_EnumToStringMap);
00302
00303 #endif /* __BE_IMAGE_NETPBM__ */
00304
```

I.41 be_image_png.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_IMAGE_PNG_H_
00012 #define BE_IMAGE_PNG_H_
00013
00014 #include <be_image_image.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Image
00019     {
00024         class PNG : public Image
00025         {
00026         public:
00027             PNG(
00028                 const uint8_t *data,
00029                 const uint64_t size,
00030                 const std::string &identifier = "",
00031                 const statusCallback_t &statusCallback =
00032                     Image::defaultStatusCallback);
00033
00034             PNG(
00035                 const Memory::uint8Array &data,
00036                 const std::string &identifier = "",
00037                 const statusCallback_t &statusCallback =
00038                     Image::defaultStatusCallback);
00039
00040             ~PNG() = default;
00041
```

```
00042             Memory::uint8Array
00043             getRawData()
00044                 const;
00045
00046             Memory::uint8Array
00047             getRawGrayscaleData(
00048                 uint8_t depth) const;
00049
00062             static bool
00063             isPNG(
00064                 const uint8_t *data,
00065                 uint64_t size);
00066         };
00067     }
00068 }
00069
00070 #endif /* BE_IMAGE_PNG_H_ */
00071
```

I.42 be_image_raw.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IMAGE_RAW_H__
00012 #define __BE_IMAGE_RAW_H__
00013
00014 #include <be_image_image.h>
00015 #include <be_memory_autoarray.h>
00016
00017 namespace BiometricEvaluation
00018 {
00019     namespace Image
00020     {
00025         class Raw : public Image {
00026         public:
00027             Raw(
00028                 const uint8_t *data,
00029                 const uint64_t size,
00030                 const Size dimensions,
00031                 const uint32_t colorDepth,
00032                 const uint16_t bitDepth,
00033                 const Resolution resolution,
00034                 const bool hasAlphaChannel,
00035                 const std::string &identifier = "",
00036                 const statusCallback_t &statusCallback =
00037                     Image::defaultStatusCallback);
00038
00039             Raw(
00040                 const BiometricEvaluation::Memory::uint8Array &data,
00041                 const Size dimensions,
00042                 const uint32_t colorDepth,
00043                 const uint16_t bitDepth,
00044                 const Resolution resolution,
00045                 const bool hasAlphaChannel,
00046                 const std::string &identifier = "",
00047                 const statusCallback_t &statusCallback =
00048                     Image::defaultStatusCallback);
00049
00050             ~Raw() = default;
00051
00052             /*
00053              * Implementations of the Image interface.
00054              */
00055
00056             Memory::uint8Array
00057             getRawData()
00058                 const;
```

```
00059
00060             Memory::uint8Array
00061             getRawGrayscaleData(
00062                 uint8_t depth) const;
00063
00064         protected:
00065
00066         private:
00067
00068         };
00069     }
00070 }
00071
00072 #endif /* __BE_IMAGE_RAW_H__ */
```

I.43 be_image_tiff.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_IMAGE_TIFF_H_
00012 #define BE_IMAGE_TIFF_H_
00013
00014 #include <be_image_image.h>
00015 #include <be_memory_indexedbuffer.h>
00016
00017 namespace BiometricEvaluation
00018 {
00019     namespace Image
00020     {
00022         class TIFF : public Image
00023         {
00024         public:
00025             TIFF(
00026                 const uint8_t *data,
00027                 const uint64_t size,
00028                 const std::string &identifier = "",
00029                 const statusCallback_t &statusCallback =
00030                     Image::defaultStatusCallback);
00031
00032             TIFF(
00033                 const Memory::uint8Array &data,
00034                 const std::string &identifier = "",
00035                 const statusCallback_t &statusCallback =
00036                     Image::defaultStatusCallback);
00037
00038             ~TIFF() = default;
00039
00040             Memory::uint8Array
00041             getRawData()
00042                 const;
00043
00044             Memory::uint8Array
00045             getRawGrayscaleData(
00046                 uint8_t depth)
00047                 const;
00048
00062             static bool
00063             isTIFF(
00064                 const uint8_t *data,
00065                 const uint64_t size);
00066
00078             static bool
00079             isTIFF(
00080                 const Memory::uint8Array &data);
00081
00096             static std::string
00097             libtiffMessageToString(
```

```
00098                 const char *module,
00099                 const char *format,
00100                 va_list args);
00101
00103             struct ClientIO
00104             {
00106                 Memory::IndexedBuffer *ib{nullptr};
00108                 const TIFF *tiffObject{nullptr};
00109             };
00110
00111         private:
00112
00124             void*
00125             getDecompressionStream()
00126                 const;
00127         };
00128     }
00129 }
00130
00131 #endif /* BE_IMAGE_TIFF_H_ */
```

I.44 be_image_wsq.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IMAGE_WSQ__
00012 #define __BE_IMAGE_WSQ__
00013
00014 #include <be_image_image.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Image
00019     {
00024         class WSQ : public Image
00025         {
00026         public:
00027             WSQ(
00028                 const uint8_t *data,
00029                 const uint64_t size,
00030                 const std::string &identifier = "",
00031                 const statusCallback_t &statusCallback =
00032                     Image::defaultStatusCallback);
00033
00034             WSQ(
00035                 const Memory::uint8Array &data,
00036                 const std::string &identifier = "",
00037                 const statusCallback_t &statusCallback =
00038                     Image::defaultStatusCallback);
00039
00040             ~WSQ() = default;
00041
00042             Memory::uint8Array
00043             getRawData()
00044                 const;
00045
00046             Memory::uint8Array
00047             getRawGrayscaleData(
00048                 uint8_t depth) const;
00049
00062             static bool
00063             isWSQ(
00064                 const uint8_t *data,
00065                 uint64_t size);
00066
00067         protected:
00068
```

```
00069         private:
00070
00071         };
00072     }
00073 }
00074
00075 #endif /* __BE_IMAGE_WSQ__ */
00076
```

I.45 be_io.h

```
00001 /**********************************************************************************
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  **********************************************************************************/
00010 #ifndef __BE_IO_H__
00011 #define __BE_IO_H__
00012
00013 #include <cstdint>
00014
00015 #include <be_framework_enumeration.h>
00016
00017 /*
00018  * This file contains items that are used within the Biometric Evaluation
00019  * IO framework.
00020  */
00021 namespace BiometricEvaluation {
00022
00031     namespace IO
00032     {
00034         enum class Mode
00035         {
00041             ReadWrite = 0,
00042
00048             ReadOnly = 1
00049         };
00050     }
00051 }
00052
00053 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00054     BiometricEvaluation::IO::Mode,
00055     BE_IO_Mode_EnumToStringMap);
00056
00057 #endif  /* __BE_IO_H__ */
```

I.46 be_io_archiverecstore.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_ARCHIVERECSTORE_H__
00012 #define __BE_ARCHIVERECSTORE_H__
00013
00014 #include <be_io_recordstore.h>
00015
00016 namespace BiometricEvaluation {
00017     namespace IO {
00043         class ArchiveRecordStore : public RecordStore {
00044         public:
00046             static const std::string MANIFEST_FILE_NAME;
```

```
00048             static const std::string ARCHIVE_FILE_NAME;
00049
00064             ArchiveRecordStore(
00065                 const std::string &pathname,
00066                 const std::string &description);
00067
00082              ArchiveRecordStore(
00083                  const std::string &pathname,
00084                  IO::Mode mode = IO::Mode::ReadOnly);
00085
00089             ~ArchiveRecordStore();
00090
00091             /*
00092              * Implementations of RecordStore methods.
00093              */
00094
00095             /*
00096                          * We need the base class insert() and replace() as well
00097              * otherwise, they are hidden by the declarations below.
00098                          */
00099                         using RecordStore::insert;
00100                         using RecordStore::replace;
00101
00102             void sync() const override;
00103
00104             void insert(
00105                 const std::string &key,
00106                 const void *const data,
00107                 const uint64_t size)
00108                 override;
00109
00110             void remove(
00111                 const std::string &key)
00112                 override;
00113
00114             Memory::uint8Array read(
00115                 const std::string &key) const override;
00116
00117             uint64_t length(
00118                 const std::string &key) const override;
00119
00120             void flush(
00121                 const std::string &key) const override;
00122
00123             RecordStore::Record sequence(
00124                 int cursor = BE_RECSTORE_SEQ_NEXT)
00125                 override;
00126
00127             std::string
00128             sequenceKey(
00129                 int cursor = BE_RECSTORE_SEQ_NEXT)
00130                 override;
00131
00132             void setCursorAtKey(
00133                 const std::string &key)
00134                 override;
00135
00136             void move(
00137                 const std::string &pathname)
00138                 override;
00139
00140             uint64_t getSpaceUsed() const override;
00141             unsigned int getCount() const override;
00142             std::string getPathname() const override;
00143             std::string getDescription() const override;
00144             void changeDescription(
00145                             const std::string &description) override;
00146
00156             bool needsVacuum();
00157
00174             static bool needsVacuum(
00175                 const std::string &pathname);
00176
00191             static void vacuum(
00192                 const std::string &pathname);
00193
00201             std::string getArchiveName() const;
```

```
00202
00210             std::string getManifestName() const;
00211
00213             static const long OFFSET_RECORD_REMOVED = -1;
00214
00215             /* Prevent copying of ArchiveRecordStore objects */
00216             ArchiveRecordStore(const ArchiveRecordStore&) = delete;
00217             ArchiveRecordStore&
00218             operator=(
00219                 const ArchiveRecordStore&) = delete;
00220
00221         private:
00222             class Impl;
00223             std::unique_ptr<ArchiveRecordStore::Impl> pimpl;
00224         };
00225     }
00226 }
00227
00228 #endif /* __BE_ARCHIVERECSTORE_H__ */
00229
```

I.47 be_io_autologger.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IO_AUTOLOGGER_H__
00012 #define __BE_IO_AUTOLOGGER_H__
00013 #include <future>
00014 #include <thread>
00015 #include <pthread.h>
00016
00017 #include <functional>
00018 #include <memory>
00019 #include <mutex>
00020 #include <optional>
00021 #include <tuple>
00022
00023 #include <be_io_filelogcabinet.h>
00024
00025 namespace BiometricEvaluation {
00026     namespace IO {
00027
00039         class AutoLogger {
00040         public:
00041
00042             /*
00043              * AutoLoggers are not copyable, but are movable.
00044              */
00045             AutoLogger(AutoLogger const &) = delete;
00046             AutoLogger& operator=(AutoLogger const &) = delete;
00047             AutoLogger(AutoLogger &&);
00048             AutoLogger& operator=(AutoLogger &&);
00049
00053             AutoLogger();
00054
00064             AutoLogger(
00065                 const std::shared_ptr<IO::Logsheet> logSheet,
00066                 const std::function<std::string()> &callback);
00067
00068             virtual ~AutoLogger();
00069
00075             std::string getComment() const;
00076
00086             void setComment(std::string_view comment);
00087
00096             void addLogEntry();
00097
```

```
00121             void startAutoLogging(
00122                 std::chrono::microseconds interval);
00123
00134             void stopAutoLogging();
00135
00144             pid_t getTaskID();
00145
00146         private:
00147             void init();
00148             void moveInit(const AutoLogger &rval);
00149             void theLogger(std::chrono::microseconds interval);
00150             std::shared_ptr<IO::Logsheet> _logSheet{};
00151             std::function<std::string()> _callback{};
00152             std::shared_future<void> _myLogger{};
00153             std::shared_ptr<std::mutex> _logMutex;
00154             std::atomic<bool> _amLogging{};
00155             std::atomic<bool> _readyFlag{};
00156             std::string _comment{};
00157             mutable std::mutex _commentMutex{};
00158             pid_t _loggerTaskID{};
00159         };
00160     }
00161 }
00162 #endif /* __BE_IO_AUTOLOGGER_H__ */
```

I.48 be_io_compressedrecstore.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IO_COMPRESSEDRECSTORE_H__
00012 #define __BE_IO_COMPRESSEDRECSTORE_H__
00013
00014 #include <memory>
00015 #include <be_io_compressor.h>
00016 #include <be_io_recordstore.h>
00017
00018 namespace BiometricEvaluation
00019 {
00020     namespace IO
00021     {
00026         class CompressedRecordStore : public RecordStore
00027         {
00028         public:
00049             CompressedRecordStore(
00050                 const std::string &pathname,
00051                 const std::string &description,
00052                     const RecordStore::Kind &recordStoreType,
00053                 const std::string &compressorType);
00054
00075             CompressedRecordStore(
00076                 const std::string &pathname,
00077                 const std::string &description,
00078                     const RecordStore::Kind &recordStoreType,
00079                 const Compressor::Kind &compressorType);
00080
00095             CompressedRecordStore(
00096                 const std::string &pathname,
00097                 IO::Mode mode = IO::Mode::ReadOnly);
00098
00099             /*
00100              * Destructor.
00101              */
00102
00103             ~CompressedRecordStore();
00104
00105             /*
00106              * Implementation of the RecordStore interface.
```

```
00107              */
00108
00109             /*
00110                          * We need the base class insert() and replace() as well
00111              * otherwise, they are hidden by the declarations below.
00112                          */
00113                         using RecordStore::insert;
00114                         using RecordStore::replace;
00115
00116             uint64_t
00117             getSpaceUsed() const override;
00118             void sync() const override;
00119             unsigned int getCount() const override;
00120             std::string getPathname() const override;
00121             std::string getDescription() const override;
00122             void changeDescription(
00123                             const std::string &description) override;
00124
00125             void
00126             insert(
00127                 const std::string &key,
00128                 const void *const data,
00129                 const uint64_t size)
00130                 override;
00131
00132             void
00133             remove(
00134                 const std::string &key) override;
00135
00136             Memory::uint8Array
00137             read(
00138                 const std::string &key) const override;
00139
00140             uint64_t
00141             length(
00142                 const std::string &key) const override;
00143
00144             void
00145             flush(
00146                 const std::string &key) const override;
00147
00148             RecordStore::Record
00149             sequence(
00150                 int cursor = BE_RECSTORE_SEQ_NEXT)
00151                 override;
00152
00153             std::string
00154             sequenceKey(
00155                 int cursor = BE_RECSTORE_SEQ_NEXT)
00156                 override;
00157
00158             void
00159             setCursorAtKey(
00160                 const std::string &key)
00161                 override;
00162
00163             void
00164             move(
00165                 const std::string &pathname)
00166                 override;
00167
00178              CompressedRecordStore(
00179                 const CompressedRecordStore &rhs) = delete;
00180
00195             CompressedRecordStore&
00196             operator=(
00197                 const CompressedRecordStore &rhs) = delete;
00198
00199         private:
00200             class Impl;
00201             std::unique_ptr<CompressedRecordStore::Impl> pimpl;
00202         };
00203     }
00204 }
00205 #endif  /* __BE_IO_COMPRESSEDRECSTORE_H__ */
```

I.49 be_io_compressor.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IO_COMPRESSOR__
00012 #define __BE_IO_COMPRESSOR__
00013
00014 #include <map>
00015 #include <memory>
00016 #include <string>
00017
00018 #include <be_error_exception.h>
00019 #include <be_framework_enumeration.h>
00020 #include <be_io_properties.h>
00021 #include <be_memory_autoarray.h>
00022
00023 namespace BiometricEvaluation
00024 {
00025     namespace IO
00026     {
00032         class Compressor
00033         {
00034         public:
00036             enum class Kind {
00037                 GZIP
00038             };
00039
00046             Compressor();
00047
00063             virtual Memory::uint8Array
00064             compress(
00065                 const uint8_t *const uncompressedData,
00066                 uint64_t uncompressedDataSize)
00067                 const = 0;
00068
00082             virtual Memory::uint8Array
00083             compress(
00084                 const Memory::uint8Array &uncompressedData)
00085                 const = 0;
00086
00103             virtual void
00104             compress(
00105                 const uint8_t *const uncompressedData,
00106                 uint64_t uncompressedDataSize,
00107                 const std::string &outputFile) const = 0;
00108
00123             virtual void
00124             compress(
00125                 const Memory::uint8Array &uncompressedData,
00126                 const std::string &outputFile) const = 0;
00127
00143             virtual Memory::uint8Array
00144             compress(
00145                 const std::string &inputFile)
00146                 const = 0;
00147
00165             virtual void
00166             compress(
00167                 const std::string &inputFile,
00168                 const std::string &outputFile) const = 0;
00169
00185             virtual Memory::uint8Array
00186             decompress(
00187                 const uint8_t *const compressedData,
00188                 uint64_t compressedDataSize)
00189                 const = 0;
00190
00204             virtual Memory::uint8Array
00205             decompress(
```

```
00206                 const Memory::uint8Array &compressedData)
00207                 const = 0;
00208
00224             virtual Memory::uint8Array
00225             decompress(
00226                 const std::string &inputFile)
00227                 const = 0;
00228
00244             virtual void
00245             decompress(
00246                 const Memory::uint8Array &compressedData,
00247                 const std::string &outputFile) const = 0;
00248
00266             virtual void
00267             decompress(
00268                 const uint8_t *const compressedData,
00269                 const uint64_t compressedDataSize,
00270                 const std::string &outputFile) const = 0;
00271
00289             virtual void
00290             decompress(
00291                 const std::string &inputFile,
00292                 const std::string &outputFile) const = 0;
00293
00308             void
00309             setOption(
00310                 const std::string &optionName,
00311                 const std::string &optionValue);
00312
00327             void
00328             setOption(
00329                 const std::string &optionName,
00330                 int64_t optionValue);
00331
00342             std::string
00343             getOption(
00344                 const std::string &optionName) const;
00345
00359             int64_t
00360             getOptionAsInteger(
00361                 const std::string &optionName) const;
00362
00370             void
00371             removeOption(
00372                 const std::string &optionName);
00373
00375             virtual ~Compressor();
00376
00389             static std::shared_ptr<Compressor>
00390             createCompressor(
00391                 Compressor::Kind compressorKind = Kind::GZIP);
00392
00402             Compressor(
00403                 const Compressor &other) = delete;
00404
00418             Compressor&
00419             operator=(
00420                 const Compressor& other) = delete;
00421
00422         private:
00424             Properties _compressorOptions;
00425         };
00426     }
00427 }
00428
00429 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00430     BiometricEvaluation::IO::Compressor::Kind,
00431     BE_IO_Compressor_Kind_EnumToStringMap);
00432
00433 #endif /* __BE_IO_COMPRESSOR__ */
```

I.50 be_io_dbrecstore.h

```
00001 /******************************************************************************
00002  * This software was developed at the National Institute of Standards and
```

```
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  ******************************************************************************/
00010 #ifndef __BE_DBRECSTORE_H__
00011 #define __BE_DBRECSTORE_H__
00012
00013 #include <be_io_recordstore.h>
00014
00015 /*
00016  * This file contains the class declaration for an implementation of a
00017  * RecordStore using a on-disk database.
00018  */
00019 namespace BiometricEvaluation {
00020
00021     namespace IO {
00022
00028         class DBRecordStore : public RecordStore {
00029         public:
00030
00045             DBRecordStore(
00046                 const std::string &pathname,
00047                 const std::string &description);
00048
00063             DBRecordStore(
00064                 const std::string &pathname,
00065                 IO::Mode mode = IO::Mode::ReadOnly);
00066
00067             /*
00068              * Destructor.
00069              */
00070             ~DBRecordStore();
00071
00072             /*
00073              * Implementation of the RecordStore interface.
00074              */
00075
00076             /*
00077                          * We need the base class insert() and replace() as well
00078              * otherwise, they are hidden by the declarations below.
00079                          */
00080                         using RecordStore::insert;
00081                         using RecordStore::replace;
00082
00083             Memory::uint8Array
00084             read(
00085                 const std::string &key) const override;
00086
00087             void insert(
00088                 const std::string &key,
00089                 const void *const data,
00090                 const uint64_t size)
00091                 override;
00092
00093             void remove(
00094                 const std::string &key)
00095                 override;
00096
00097             uint64_t length(
00098                 const std::string &key) const override;
00099
00100            void flush(
00101                const std::string &key) const override;
00102
00103            RecordStore::Record sequence(
00104                int cursor = BE_RECSTORE_SEQ_NEXT)
00105                override;
00106
00107            std::string
00108            sequenceKey(
00109                int cursor = BE_RECSTORE_SEQ_NEXT)
00110                override;
00111
00112            void setCursorAtKey(
```

```
00113                 const std::string &key)
00114                 override;
00115
00116             void move(
00117                 const std::string &pathname)
00118                 override;
00119
00120             uint64_t getSpaceUsed() const override;
00121             void sync() const override;
00122             unsigned int getCount() const override;
00123             std::string getPathname() const override;
00124             std::string getDescription() const override;
00125             void changeDescription(
00126                             const std::string &description) override;
00127
00128             /* Prevent copying of DBRecordStore objects */
00129             DBRecordStore(const DBRecordStore&) = delete;
00130             DBRecordStore& operator=(const DBRecordStore&) = delete;
00131
00132         private:
00133             class Impl;
00134             std::unique_ptr<DBRecordStore::Impl> pimpl;
00135         };
00136     }
00137 }
00138 #endif  /* __BE_DBRECSTORE_H__ */
```

I.51 be_io_filelogcabinet.h

```
00001 /**************************************************************************
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  *************************************************************************/
00010 #ifndef __BE_IO_LOGCABINET_H
00011 #define __BE_IO_LOGCABINET_H
00012
00013
00014 #include <cstdint>
00015 #include <memory>
00016 #include <sstream>
00017 #include <string>
00018 #include <vector>
00019
00020 #include <be_error_exception.h>
00021 #include <be_io_filelogsheet.h>
00022
00023 namespace BiometricEvaluation {
00024     namespace IO {
00033     class FileLogCabinet {
00034         public:
00035
00049             FileLogCabinet(
00050                 const std::string &pathname,
00051                 const std::string &description);
00052
00064             FileLogCabinet(
00065                 const std::string &pathname);
00066
00067             ~FileLogCabinet();
00068
00088             std::shared_ptr<FileLogsheet> newLogsheet(
00089                 const std::string &name,
00090                 const std::string &description);
00091
00098             std::string getPathname();
00099
00106             std::string getDescription();
00107
00114             unsigned int getCount();
00115
```

```
00116         private:
00117
00118             /* The directory where the cabinet is rooted */
00119             std::string _pathname;
00120
00121             /* A textual description of the cabinet. */
00122             std::string _description;
00123
00124             /* Number of items in the cabinet */
00125             unsigned int _count;
00126
00127             /* The current record position cursor */
00128             int _cursor;
00129
00130             /*
00131              * Return the full path of a file stored as part
00132              * of the FileLogCabinet, typically _pathname + name.
00133              */
00134             std::string canonicalName(const std::string &name);
00135
00136             /* Read the contents of the common control file format
00137              * for all FileLogCabinet.
00138              */
00139             void readControlFile();
00140
00141             /* Write the contents of the common control file format
00142              * for all FileLogCabinet.
00143              */
00144             void writeControlFile();
00145
00146         private:
00147             /* Prevent copying of FileLogCabinet objects */
00148             FileLogCabinet(const FileLogCabinet&);
00149             FileLogCabinet& operator=(const FileLogCabinet&);
00150     };
00151     }
00152 }
00153 #endif  /* __BE_IO_LOGCABINET_H */
```

I.52 be_io_filelogsheet.h

```
00001
00011 #ifndef __BE_IO_FILELOGSHEET_H__
00012 #define __BE_IO_FILELOGSHEET_H__
00013
00014 #include <fstream>
00015 #include <vector>
00016
00017 #include <be_io_logsheet.h>
00018
00019 namespace BiometricEvaluation
00020 {
00021     namespace IO
00022     {
00034         class FileLogsheet : public IO::Logsheet
00035         {
00036         public:
00037
00061             FileLogsheet(
00062                 const std::string &url,
00063                 const std::string &description);
00064
00086             FileLogsheet(
00087                 const std::string &url);
00088
00090             ~FileLogsheet();
00091
00107             static void
00108             mergeLogsheets(
00109                 std::vector<std::shared_ptr<FileLogsheet>>
00110                 &logsheets);
00111
00113             static const int32_t BE_FILELOGSHEET_SEQ_START = 1;
00115             static const int32_t BE_FILELOGSHEET_SEQ_NEXT = 2;
00116
```

```
00142             std::string
00143             sequence(
00144                 bool allEntries = false,
00145                 bool trim = true,
00146                 int32_t cursor = BE_FILELOGSHEET_SEQ_NEXT);
00147
00160             static std::string
00161             trim(
00162                 const std::string &entry);
00163
00164
00165             /* Declare implementations of parent interface */
00166             void write(const std::string &entry);
00167             void writeComment(const std::string &entry);
00168             void writeDebug(const std::string &entry);
00169             void sync();
00170
00171         protected:
00173             FileLogsheet(const FileLogsheet&);
00175             FileLogsheet& operator=(const FileLogsheet&);
00176
00178             std::unique_ptr<std::fstream> _theLogFile;
00179
00187             void
00188             updateCursor();
00189
00191             std::shared_ptr<std::fstream> _sequenceFile;
00192
00194             std::streamoff _cursor;
00195         };
00196     }
00197 }
00198
00199 #endif  /* __BE_IO_FILELOGSHEET_H__ */
```

I.53 be_io_filerecstore.h

```
00001 /**********************************************************************************
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  **********************************************************************************/
00010 #ifndef __BE_FILERECSTORE_H__
00011 #define __BE_FILERECSTORE_H__
00012
00013 #include <be_io_recordstore.h>
00014
00015 /*
00016  * This file contains the class declaration for an implementation of a
00017  * RecordStore using a on-disk database.
00018  */
00019 namespace BiometricEvaluation {
00020
00021     namespace IO {
00022
00034         class FileRecordStore : public RecordStore {
00035         public:
00036
00050             FileRecordStore(
00051                 const std::string &pathname,
00052                 const std::string &description);
00053
00068             FileRecordStore(
00069                 const std::string &name,
00070                 IO::Mode mode = IO::Mode::ReadOnly);
00071
00072             ~FileRecordStore();
00073
00074             /*
00075              * Methods that implement the RecordStore interface.
00076              */
```

```
00077
00078             /*
00079                          * We need the base class insert() and replace() as well
00080              * otherwise, they are hidden by the declarations below.
00081                          */
00082                         using RecordStore::insert;
00083                         using RecordStore::replace;
00084
00085             void insert(
00086                 const std::string &key,
00087                 const void *const data,
00088                 const uint64_t size)
00089                 override;
00090
00091             void remove(
00092                 const std::string &key)
00093                 override;
00094
00095             Memory::uint8Array read(
00096                 const std::string &key) const override;
00097
00098             void replace(
00099                 const std::string &key,
00100                 const void *const data,
00101                 const uint64_t size) override final;
00102
00103             uint64_t length(
00104                 const std::string &key) const override;
00105
00106             void flush(
00107                 const std::string &key) const override;
00108
00109             RecordStore::Record sequence(
00110                 int cursor = BE_RECSTORE_SEQ_NEXT)
00111                 override;
00112
00113             std::string
00114             sequenceKey(
00115                 int cursor = BE_RECSTORE_SEQ_NEXT)
00116                 override;
00117
00118             void setCursorAtKey(
00119                 const std::string &key)
00120                 override;
00121
00122             void move(
00123                 const std::string &pathname)
00124                 override;
00125
00126             uint64_t getSpaceUsed() const override;
00127             void sync() const override;
00128             unsigned int getCount() const override;
00129             std::string getPathname() const override;
00130             std::string getDescription() const override;
00131             void changeDescription(
00132                 const std::string &description) override;
00133
00134             /* Prevent copying of FileRecordStore objects */
00135             FileRecordStore(const FileRecordStore&) = delete;
00136             FileRecordStore& operator=(const FileRecordStore&) =
00137                 delete;
00138         protected:
00139         private:
00140             class Impl;
00141             std::unique_ptr<FileRecordStore::Impl> pimpl;
00142         };
00143     }
00144 }
00145 #endif  /* __BE_FILERECSTORE_H__ */
```

I.54 be_io_gzip.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
```

```
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IO_GZIP__
00012 #define __BE_IO_GZIP__
00013
00014 #include <string>
00015 #include <zlib.h>
00016
00017 #include <be_error_exception.h>
00018 #include <be_io_compressor.h>
00019 #include <be_io_properties.h>
00020 #include <be_memory_autoarray.h>
00021
00022 namespace BiometricEvaluation
00023 {
00024     namespace IO
00025     {
00030         class GZip : public Compressor
00031         {
00032         public:
00033             /*
00034              * zlib compressor property keys.
00035              */
00037             static const std::string COMPRESSION_LEVEL;
00039             static const std::string COMPRESSION_STRATEGY;
00041             static const std::string COMPRESSION_METHOD;
00043             static const std::string INPUT_DATA_TYPE;
00045             static const std::string WINDOW_BITS;
00047             static const std::string MEMORY_LEVEL;
00049             static const std::string CHUNK_SIZE;
00050
00051             GZip();
00052
00053             Memory::uint8Array
00054             compress(
00055                 const uint8_t *const uncompressedData,
00056                 uint64_t uncompressedDataSize)
00057                 const;
00058
00059             Memory::uint8Array
00060             compress(
00061                 const Memory::uint8Array &uncompressedData)
00062                 const;
00063
00064             void
00065             compress(
00066                 const uint8_t *const uncompressedData,
00067                 uint64_t uncompressedDataSize,
00068                 const std::string &outputFile) const;
00069
00070             void
00071             compress(
00072                 const Memory::uint8Array &uncompressedData,
00073                 const std::string &outputFile) const;
00074
00075             Memory::uint8Array
00076             compress(
00077                 const std::string &inputFile)
00078                 const;
00079
00080             void
00081             compress(
00082                 const std::string &inputFile,
00083                 const std::string &outputFile) const;
00084
00085             Memory::uint8Array
00086             decompress(
00087                 const uint8_t *const compressedData,
00088                 uint64_t compressedDataSize)
00089                 const;
00090
00091             Memory::uint8Array
```

```
00092             decompress(
00093                 const Memory::uint8Array &compressedData)
00094                 const;
00095
00096             Memory::uint8Array
00097             decompress(
00098                 const std::string &input)
00099                 const;
00100
00101             void
00102             decompress(
00103                 const std::string &inputFile,
00104                 const std::string &outputFile) const;
00105
00106             void
00107             decompress(
00108                 const uint8_t *const compressedData,
00109                 const uint64_t compressedDataSize,
00110                 const std::string &outputFile) const;
00111
00112             void
00113             decompress(
00114                 const Memory::uint8Array &compressedData,
00115                 const std::string &outputFile) const;
00116
00117             ~GZip();
00118
00129             GZip(
00130                 const GZip &other) = delete;
00131
00145             GZip&
00146             operator=(
00147                 const GZip& other) = delete;
00148
00149         private:
00157             z_stream
00158             initCompressionStream()
00159                 const;
00160
00185             int32_t
00186             compressChunk(
00187                 uint8_t flush,
00188                 uint64_t chunkSize,
00189                 uint64_t &totalCompressedBytes,
00190                 Memory::uint8Array &compressedBuf,
00191                 bool compressedBufIsChunk,
00192                 z_stream &strm) const;
00193
00201             z_stream
00202             initDecompressionStream()
00203                 const;
00204
00227             int32_t
00228             decompressChunk(
00229                 uint64_t chunkSize,
00230                 uint64_t &totalUncompressedBytes,
00231                 Memory::uint8Array &uncompressedBuf,
00232                 bool uncompressedBufIsChunk,
00233                 z_stream &strm) const;
00234
00236             static const uint8_t GZIP_WBITS_MAGIC = 16;
00237         };
00238     }
00239 }
00240
00241 #endif /* __BE_IO_GZIP__ */
```

I.55 be_io_listrecstore.h

```
00001
00011 #ifndef __BE_IO_LISTRECSTORE_H__
00012 #define __BE_IO_LISTRECSTORE_H__
00013
00014 #include <memory.h>
00015 #include <be_io_recordstore.h>
```

```
00016
00017 namespace BiometricEvaluation
00018 {
00019     namespace IO
00020     {
00057         class ListRecordStore : public RecordStore {
00058         public:
00060             ListRecordStore(
00061                 const std::string &pathname);
00062
00064             ~ListRecordStore() = default;
00065
00066             /*
00067              * Implementation of the RecordStore interface.
00068              */
00069
00070             /*
00071                          * We need the base class insert() and replace() as well
00072              * otherwise, they are hidden by the declarations below.
00073                          */
00074                         using RecordStore::insert;
00075                         using RecordStore::replace;
00076
00077             void
00078             insert(
00079                 const std::string &key,
00080                 const void *const data,
00081                 const uint64_t size)
00082                 override;
00083
00084             void
00085             remove(
00086                 const std::string &key)
00087                 override;
00088
00089             Memory::uint8Array
00090             read(
00091                 const std::string &key) const override;
00092
00093             void
00094             replace(
00095                 const std::string &key,
00096                 const void *const data,
00097                 const uint64_t size) override final;
00098
00099             uint64_t
00100             length(
00101                 const std::string &key) const override;
00102
00103             void
00104             flush(
00105                 const std::string &key) const override;
00106
00107                 void
00108             sync() const override;
00109
00110             RecordStore::Record
00111             sequence(
00112                 int cursor = BE_RECSTORE_SEQ_NEXT)
00113                 override;
00114
00115             std::string
00116             sequenceKey(
00117                 int cursor = BE_RECSTORE_SEQ_NEXT)
00118                 override;
00119
00120             void
00121             setCursorAtKey(
00122                 const std::string &key)
00123                 override;
00124
00125             void
00126             move(
00127                 const std::string &pathname)
00128                 override;
00129
00130             uint64_t
```

```
00131             getSpaceUsed() const
00132             override;
00133
00134             unsigned int getCount() const override;
00135             std::string getPathname() const override;
00136             std::string getDescription() const override;
00137             void changeDescription(
00138                             const std::string &description) override;
00139
00140         private:
00141             class Impl;
00142             std::unique_ptr<ListRecordStore::Impl> pimpl;
00143         };
00144     }
00145 }
00146
00147 #endif /* __BE_IO_LISTRECSTORE_H__ */
```

I.56 be_io_logsheet.h

```
00001
00011 #ifndef __BE_IO_LOGSHEET_H__
00012 #define __BE_IO_LOGSHEET_H__
00013
00014 #include <cstdint>
00015 #include <memory>
00016 #include <sstream>
00017 #include <string>
00018
00019 namespace BiometricEvaluation
00020 {
00021     namespace IO
00022     {
00057         class Logsheet : public std::ostream
00058         {
00059         public:
00060             enum class Kind {
00062                 Null,
00064                 File,
00066                 Syslog
00067             };
00068
00070             static const char CommentDelimiter = '#';
00071
00073             static const char EntryDelimiter = 'E';
00074
00076             static const char DebugDelimiter = 'D';
00077
00079             static const std::string DescriptionTag;
00080
00085             static const std::string FILEURLSCHEME;
00086
00091             static const std::string SYSLOGURLSCHEME;
00092
00105             static Logsheet::Kind getTypeFromURL(
00106                 const std::string &url);
00107
00114             Logsheet();
00115
00126             static bool
00127             lineIsEntry(const std::string &line);
00128
00140             static bool
00141             lineIsComment(const std::string &line);
00142
00154             static bool
00155             lineIsDebug(const std::string &line);
00156
00158             virtual ~Logsheet();
00159
00173             void
00174             newEntry();
00175
00184             std::string
00185             getCurrentEntry() const;
```

```
00186
00188             void
00189             resetCurrentEntry();
00190
00198             uint32_t
00199             getCurrentEntryNumber() const;
00200
00215             virtual void
00216             write(
00217                 const std::string &entry);
00218
00235             virtual void
00236             writeComment(
00237                 const std::string &entry);
00238
00254             virtual void
00255             writeDebug(const std::string &entry);
00256
00270             void
00271             setCommit(const bool state);
00272
00281             bool
00282             getCommit() const;
00283
00297             void
00298             setDebugCommit(const bool state);
00299
00308             bool
00309             getDebugCommit() const;
00310
00324             void
00325             setCommentCommit(const bool state);
00326
00335             bool
00336             getCommentCommit() const;
00337
00350             virtual void
00351             sync();
00352
00363             void
00364             setAutoSync(bool state);
00365
00372             bool
00373             getAutoSync() const;
00374
00387             static std::string
00388             trim(
00389                 const std::string &entry);
00390
00391         protected:
00396             void
00397             incrementEntryNumber();
00398
00406             std::string
00407             getCurrentEntryNumberAsString() const;
00408
00409         private:
00411             std::stringbuf _sbuf{};
00412
00414             uint32_t _entryNumber{};
00415
00417             bool _autoSync{};
00418
00420             bool _commit{};
00421
00423             bool _debugCommit{};
00424
00426             bool _commentCommit{};
00427         };
00428     }
00429 }
00430
00431 #endif  /* __BE_IO_LOGSHEET_H__ */
```

I.57 be_io_persistentrecordstoreunion.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_IO_PERSISTENTRECORDSTOREUNION_H_
00012 #define BE_IO_PERSISTENTRECORDSTOREUNION_H_
00013
00014 #include <be_io_recordstoreunion.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace IO
00019     {
00028         class PersistentRecordStoreUnion : public RecordStoreUnion
00029         {
00030         public:
00038             PersistentRecordStoreUnion(
00039                 const std::string &path);
00040
00050             PersistentRecordStoreUnion(
00051                 const std::string &path,
00052                 const std::map<const std::string, const std::string>
00053                 &recordStores);
00054
00066             PersistentRecordStoreUnion(
00067                 const std::string &path,
00068                 std::initializer_list<std::pair<const std::string,
00069                 const std::string>> &recordStores);
00070
00072             ~PersistentRecordStoreUnion() = default;
00073
00074         protected:
00076             class Impl;
00077         };
00078     }
00079 }
00080
00081 #endif /* BE_IO_PERSISTENTRECORDSTOREUNION_H_ */
```

I.58 be_io_properties.h

```
00001 /******************************************************************************
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  *****************************************************************************/
00010 #ifndef __BE_IO_PROPERTIES_H__
00011 #define __BE_IO_PROPERTIES_H__
00012
00013 #include <map>
00014 #include <string>
00015 #include <vector>
00016
00017 #include <be_error_exception.h>
00018 #include <be_io.h>
00019 #include <be_memory_autoarray.h>
00020
00021 namespace BiometricEvaluation
00022 {
00023     namespace IO
00024     {
00030         class Properties {
```

```
00031         public:
00041             Properties(
00042                 IO::Mode mode = IO::Mode::ReadWrite,
00043                 const std::map<std::string, std::string>
00044                 &defaults = {});
00045
00066             Properties(
00067                 const uint8_t *buffer,
00068                 const size_t size,
00069                 IO::Mode mode = IO::Mode::ReadWrite,
00070                 const std::map<std::string, std::string>
00071                 &defaults = {});
00072
00091             virtual void
00092             setProperty(
00093                 const std::string &property,
00094                 const std::string &value);
00095
00113             virtual void
00114             setPropertyFromInteger(
00115                 const std::string &property,
00116                 int64_t value);
00117
00135             virtual void
00136             setPropertyFromDouble(
00137                 const std::string &property,
00138                 double value);
00139
00157             virtual void
00158             setPropertyFromBoolean(
00159                 const std::string &property,
00160                 bool value);
00161
00174             virtual void
00175             removeProperty(
00176                 const std::string &property);
00177
00188             virtual std::string
00189             getProperty(
00190                 const std::string &property) const;
00191
00210             virtual int64_t
00211             getPropertyAsInteger(
00212                 const std::string &property) const;
00213
00228             virtual double
00229             getPropertyAsDouble(
00230                 const std::string &property) const;
00231
00246             virtual bool
00247             getPropertyAsBoolean(
00248                 const std::string &property)
00249                 const;
00250
00258             std::vector<std::string>
00259             getPropertyKeys() const;
00260
00262             virtual ~Properties();
00263
00264         protected:
00272             BiometricEvaluation::IO::Mode
00273             getMode()
00274                 const;
00275
00292             void
00293             initWithBuffer(
00294                 const Memory::uint8Array &buffer,
00295                 const std::map<std::string, std::string> &defaults);
00296
00315             void
00316             initWithBuffer(
00317                 const uint8_t *const buffer,
00318                 size_t size,
00319                 const std::map<std::string, std::string> &defaults);
00320
00321         private:
00326             using PropertiesMap = std::map<std::string, std::string>;
```

```
00327
00329             PropertiesMap _properties;
00330
00332             IO::Mode _mode;
00333
00341             void
00342             registerDefaults(
00343                 const PropertiesMap &defaults);
00344         };
00345     }
00346 }
00347 #endif  /* __BE_IO_PROPERTIES_H__ */
```

I.59 be_io_propertiesfile.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IO_PROPERTIESFILE_H__
00012 #define __BE_IO_PROPERTIESFILE_H__
00013
00014 #include <map>
00015 #include <string>
00016
00017 #include <be_io.h>
00018 #include <be_io_properties.h>
00019 #include <be_error_exception.h>
00020
00021 namespace BiometricEvaluation
00022 {
00023     namespace IO
00024     {
00049         class PropertiesFile : public Properties
00050         {
00051         public:
00071             PropertiesFile(
00072                 const std::string &pathname,
00073                 IO::Mode mode = IO::Mode::ReadOnly,
00074                 const std::map<std::string, std::string>
00075                 &defaults = {});
00076
00089             void
00090             sync();
00091
00109             void
00110             changeName(
00111                 const std::string &pathname);
00112
00114             ~PropertiesFile();
00115
00126             PropertiesFile(
00127                 const PropertiesFile &other) = delete;
00128
00143             PropertiesFile&
00144             operator=(
00145                 const PropertiesFile &other) = delete;
00146
00147         private:
00149             std::string _pathname;
00150
00158             void
00159             initPropertiesFile(
00160                 const std::map<std::string, std::string> &defaults);
00161         };
00162     }
00163 }
00164 #endif  /* __BE_IO_PROPERTIESFILE_H__ */
```

I.60 be_io_recordstore.h

```
00001 /**************************************************************************
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  *************************************************************************/
00010 #ifndef __BE_IO_RECORDSTORE_H__
00011 #define __BE_IO_RECORDSTORE_H__
00012
00013 #include <functional>
00014 #include <memory>
00015 #include <string>
00016 #include <vector>
00017
00018 #include <be_framework_enumeration.h>
00019 #include <be_io.h>
00020 #include <be_memory_autoarray.h>
00021
00022 /*
00023  * This file contains the class declaration for the RecordStore, a virtual
00024  * class that represents a collection of named blobs of data.
00025  */
00026
00027 namespace BiometricEvaluation {
00028
00029     namespace IO {
00030         class RecordStoreIterator;
00031
00051         class RecordStore {
00052         public:
00053             struct Record {
00057                 Record();
00058
00067                 Record(
00068                     const std::string &key,
00069                     const Memory::uint8Array &data);
00070                 std::string key;
00071                 Memory::uint8Array data;
00072             };
00073             using Record = struct Record;
00074
00075             using iterator = IO::RecordStoreIterator;
00076
00078             enum class Kind
00079             {
00081                 BerkeleyDB,
00083                 Archive,
00085                 File,
00087                 SQLite,
00089                 Compressed,
00091                 List,
00092
00094                 Default = BerkeleyDB
00095             };
00096
00101             static const std::string INVALIDKEYCHARS;
00102
00103             virtual ~RecordStore();
00104
00110             virtual std::string getDescription() const = 0;
00111
00117             virtual unsigned int getCount() const = 0;
00118
00125             virtual std::string getPathname() const = 0;
00126
00139             virtual void move(
00140                 const std::string &pathname) = 0;
00141
00150             virtual void changeDescription(
00151                 const std::string &description) = 0;
00152
00169             virtual uint64_t getSpaceUsed() const = 0;
```

```
00170
00179             virtual void sync() const = 0;
00180
00197             void
00198             virtual insert(
00199                 const std::string &key,
00200                 const Memory::uint8Array &data);
00201
00220             virtual void
00221             insert(
00222                 const std::string &key,
00223                 const void *const data,
00224                 const uint64_t size) = 0;
00225
00237             virtual void remove(
00238                 const std::string &key) = 0;
00239
00257             virtual Memory::uint8Array
00258             read(
00259                 const std::string &key) const = 0;
00260
00276             virtual void replace(
00277                 const std::string &key,
00278                 const Memory::uint8Array &data);
00279
00297             virtual void replace(
00298                 const std::string &key,
00299                 const void *const data,
00300                 const uint64_t size);
00301
00315             virtual uint64_t length(
00316                 const std::string &key) const = 0;
00317
00328             virtual void flush(
00329                 const std::string &key) const = 0;
00330
00332             static const int BE_RECSTORE_SEQ_START = 1;
00334             static const int BE_RECSTORE_SEQ_NEXT = 2;
00335
00361             virtual RecordStore::Record
00362             sequence(
00363                 int cursor = BE_RECSTORE_SEQ_NEXT) = 0;
00364
00389             virtual std::string
00390             sequenceKey(
00391                 int cursor = BE_RECSTORE_SEQ_NEXT) = 0;
00392
00409             virtual void setCursorAtKey(
00410                 const std::string &key) = 0;
00411
00424             virtual bool
00425             containsKey(
00426                 const std::string &key)
00427                 const;
00428
00430             virtual iterator
00431             begin()
00432                 noexcept;
00433
00435             virtual iterator
00436             end()
00437                 noexcept;
00438
00450             static bool
00451             isRecordStore(
00452                 const std::string &pathname);
00453
00480             static std::shared_ptr<RecordStore> openRecordStore(
00481                 const std::string &pathname,
00482                 IO::Mode mode = Mode::ReadOnly);
00483
00508             static std::shared_ptr<RecordStore> createRecordStore(
00509                 const std::string &pathname,
00510                 const std::string &description,
00511                 const IO::RecordStore::Kind &kind);
00512
00525             static void removeRecordStore(
```

```
00526                const std::string &pathname);
00527
00554            static void mergeRecordStores(
00555                const std::string &mergePathname,
00556                const std::string &description,
00557                const IO::RecordStore::Kind &kind,
00558                const std::vector<std::string> &pathnames,
00559                const std::function<bool()> &interrupt =
00560                []() {return (false);});
00561
00562            class Impl;
00563        protected:
00564        private:
00565        };
00566
00579        class RecordStoreIterator
00580        {
00581        public:
00582            /*
00583             * Satisfy std::iterator_traits<> expectations.
00584             */
00585
00587            using iterator_category = std::forward_iterator_tag;
00589            using value_type = RecordStore::Record;
00591            using difference_type = std::ptrdiff_t;
00593            using pointer = value_type*;
00595            using reference = value_type&;
00596
00606            RecordStoreIterator() = default;
00607
00624            RecordStoreIterator(
00625                IO::RecordStore *recordStore,
00626                bool atEnd);
00627
00629            RecordStoreIterator(
00630                const RecordStoreIterator &rhs) = default;
00632            RecordStoreIterator(
00633                RecordStoreIterator &&rvalue) = default;
00635            ~RecordStoreIterator() = default;
00636
00637            /*
00638             * Operators.
00639             */
00640
00642            reference
00643            operator*();
00644
00646            pointer
00647            operator->();
00648
00650            RecordStoreIterator&
00651            operator++();
00652
00654            RecordStoreIterator
00655            operator++(
00656                int);
00657
00668            RecordStoreIterator
00669            operator+=(
00670                difference_type rhs);
00671
00682            RecordStoreIterator
00683            operator+(
00684                difference_type rhs);
00685
00696            bool
00697            operator==(
00698                const RecordStoreIterator &rhs);
00699
00714            inline bool
00715            operator!=(
00716                const RecordStoreIterator &rhs)
00717            {
00718                return (!(*this == rhs));
00719            }
00720
00721            /* Default copy assignment operator */
```

```
00722             RecordStoreIterator&
00723             operator=(
00724                 const RecordStoreIterator &rhs) = default;
00725
00727             RecordStoreIterator&
00728             operator=(
00729                 RecordStoreIterator &&rhs) = default;
00730
00731         private:
00733             IO::RecordStore *_recordStore{nullptr};
00735             bool _atEnd{true};
00737             value_type _currentRecord{};
00738
00740             void
00741             setBegin();
00742
00750             void
00751             step(
00752                 difference_type steps = 1);
00753
00755             void
00756             setEnd();
00757         };
00758     }
00759 }
00760
00761 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00762     BiometricEvaluation::IO::RecordStore::Kind,
00763     BE_IO_RecordStore_Kind_EnumToStringMap);
00764
00765 #endif  /* __BE_IO_RECORDSTORE_H__ */
```

I.61 be_io_recordstoreunion.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_IO_RECORDSTOREUNION_H_
00012 #define BE_IO_RECORDSTOREUNION_H_
00013
00014 #include <map>
00015 #include <memory>
00016 #include <string>
00017 #include <utility>
00018 #include <vector>
00019
00020 #include <be_io.h>
00021 #include <be_io_recordstore.h>
00022 #include <be_memory_autoarray.h>
00023
00024 namespace BiometricEvaluation
00025 {
00026     namespace IO
00027     {
00036         class RecordStoreUnion
00037         {
00038         public:
00046             RecordStoreUnion(
00047                 const std::map<const std::string, const std::string>
00048                 &recordStores);
00049
00060             RecordStoreUnion(
00061                 std::map<const std::string,
00062                 const std::string>::iterator first,
00063                 std::map<const std::string,
00064                 const std::string>::iterator last);
00065
00073             RecordStoreUnion(
```

```
00074                 std::initializer_list<std::pair<
00075                 const std::string, const std::string>>
00076                 recordStores);
00077
00089             RecordStoreUnion(
00090                 const std::map<const std::string,
00091                 const std::shared_ptr<
00092                 BiometricEvaluation::IO::RecordStore>>
00093                 &recordStores);
00094
00109             RecordStoreUnion(
00110                 std::map<const std::string, const std::shared_ptr<
00111                 BiometricEvaluation::IO::RecordStore>>::iterator
00112                 first,
00113                 std::map<const std::string, const std::shared_ptr<
00114                 BiometricEvaluation::IO::RecordStore>>::iterator
00115                 last);
00116
00128             RecordStoreUnion(
00129                 std::initializer_list<std::pair<const std::string,
00130                 const std::shared_ptr<
00131                 BiometricEvaluation::IO::RecordStore>>>
00132                 recordStores);
00133
00144             std::shared_ptr<BiometricEvaluation::IO::RecordStore>
00145             getRecordStore(
00146                 const std::string &name)
00147                 const;
00148
00157             std::vector<std::string>
00158             getNames()
00159                 const;
00160
00161             /*
00162              * RecordStore Operations.
00163              */
00164
00186             std::map<const std::string,
00187             BiometricEvaluation::Memory::uint8Array>
00188             read(
00189                 const std::string &key)
00190                 const;
00191
00214             std::map<const std::string, uint64_t>
00215             length(
00216                 const std::string &key)
00217                 const;
00218
00219             /* Prevent copying of RecordStoreUnion objects */
00220             RecordStoreUnion(const RecordStoreUnion&) = delete;
00221             RecordStoreUnion& operator=(const RecordStoreUnion&)
00222                 = delete;
00223
00225             ~RecordStoreUnion();
00226
00227         protected:
00229             class Impl;
00230
00241             RecordStoreUnion();
00242
00250             void
00251             setImpl(const
00252                 std::shared_ptr<RecordStoreUnion::Impl> &pimpl);
00253
00254         private:
00256             std::shared_ptr<RecordStoreUnion::Impl> pimpl;
00257         };
00258     }
00259 }
00260
00261 #endif /* BE_IO_RECORDSTOREUNION_H_ */
```

I.62 be_io_sqliterecstore.h

```
00001 /*
```

```
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IO_SQLITERECORDSTORE_H__
00012 #define __BE_IO_SQLITERECORDSTORE_H__
00013
00014 #include <be_io_recordstore.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace IO
00019     {
00025         class SQLiteRecordStore : public RecordStore
00026         {
00027         public:
00028             SQLiteRecordStore(
00029                 const std::string &pathname,
00030                 const std::string &description);
00031
00032             SQLiteRecordStore(
00033                 const std::string &pathname,
00034                 IO::Mode mode = Mode::ReadOnly);
00035
00036             /*
00037                          * We need the base class insert() and replace() as well
00038              * otherwise, they are hidden by the declarations below.
00039                          */
00040                         using RecordStore::insert;
00041                         using RecordStore::replace;
00042
00043             void
00044             move(
00045                 const std::string &pathname)
00046                 override;
00047
00048             void sync() const override;
00049             unsigned int getCount() const override;
00050             std::string getPathname() const override;
00051             std::string getDescription() const override;
00052
00053             void
00054             changeDescription(
00055                 const std::string &description)
00056                 override;
00057
00058             uint64_t
00059             getSpaceUsed() const override;
00060
00061             void
00062             insert(
00063                 const std::string &key,
00064                 const void *const data,
00065                 const uint64_t size)
00066                 override;
00067
00068             void
00069             remove(
00070                 const std::string &key)
00071                 override;
00072
00073             Memory::uint8Array
00074             read(
00075                 const std::string &key) const override;
00076
00077             uint64_t
00078             length(
00079                 const std::string &key) const override;
00080
00081             void
00082             flush(
00083                 const std::string &key) const override;
```

```
00084
00085             RecordStore::Record
00086             sequence(
00087                 int cursor = BE_RECSTORE_SEQ_NEXT)
00088                 override;
00089
00090             std::string
00091             sequenceKey(
00092                 int cursor = BE_RECSTORE_SEQ_NEXT)
00093                 override;
00094
00095             void
00096             setCursorAtKey(
00097                 const std::string &key)
00098                 override;
00099
00100             ~SQLiteRecordStore();
00101
00102             SQLiteRecordStore(const SQLiteRecordStore&) = delete;
00103             SQLiteRecordStore&
00104             operator=(
00105                 const SQLiteRecordStore&) = delete;
00106
00107         protected:
00108         private:
00109             class Impl;
00110             std::unique_ptr<SQLiteRecordStore::Impl> pimpl;
00111         };
00112     }
00113 }
00114 #endif  /* __BE_IO_SQLITERECORDSTORE_H__ */
```

I.63 be_io_syslogsheet.h

```
00001
00011 #ifndef __BE_IO_SYSLOGSHEET_H__
00012 #define __BE_IO_SYSLOGSHEET_H__
00013 #endif
00014
00015 #include <be_io_logsheet.h>
00016
00017 namespace BiometricEvaluation
00018 {
00019     namespace IO
00020     {
00045         class SysLogsheet : public IO::Logsheet
00046         {
00047         public:
00073             SysLogsheet(
00074                 const std::string &url,
00075                 const std::string &description,
00076                 const std::string &appname,
00077                 bool sequenced,
00078                 bool utc);
00079
00108             SysLogsheet(
00109                 const std::string &url,
00110                 const std::string &description,
00111                 const std::string &appname,
00112                 const std::string &hostname,
00113                 bool sequenced,
00114                 bool utc);
00115
00117             ~SysLogsheet();
00118
00119             /* Declare implementations of parent interface */
00120             void
00121             write(const std::string &entry);
00122             void
00123             writeComment(const std::string &entry);
00124             void
00125             writeDebug(const std::string &entry);
00126             void
00127             sync();
00128
```

```
00129         protected:
00131             SysLogsheet(const SysLogsheet&);
00132
00134             SysLogsheet& operator=(const SysLogsheet&);
00135
00137             void setup(
00138                 const std::string &url,
00139                 const std::string &description);
00140
00142             void writeToLogger(
00143                 const std::string &priority,
00144                 const char delimiter,
00145                 const std::string &prefix,
00146                 const std::string &message);
00147
00148             std::string _hostname;
00149             std::string _appname;
00150             std::string _procid;
00151
00153             int _sockFD;
00154
00156             bool _sequenced;
00157
00159             bool _operational;
00160
00162             bool _utc;
00163         };
00164     }
00165 }
00166
```

I.64 be_io_utility.h

```
00001 /*****************************************************************************
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  *****************************************************************************/
00010 #ifndef __BE_IO_UTILITY_H
00011 #define __BE_IO_UTILITY_H
00012
00013 #include <sys/stat.h>
00014
00015 #include <cstdint>
00016 #include <fstream>
00017 #include <string>
00018
00019 #include <be_error_exception.h>
00020 #include <be_memory_autoarray.h>
00021 #include <be_sysdeps.h>
00022
00023 namespace BiometricEvaluation
00024 {
00025     namespace IO
00026     {
00031         namespace Utility {
00032
00050             void removeDirectory(
00051                 const std::string &directory,
00052                 const std::string &prefix);
00053
00067             void removeDirectory(
00068                 const std::string &pathname);
00069
00091             void copyDirectoryContents(
00092                 const std::string &sourcepath,
00093                 const std::string &targetpath,
00094                 const bool removesource = false);
00095
00118             void setAsideName(
00119                 const std::string &name);
```

```
00120
00135           uint64_t getFileSize(
00136               const std::string &pathname);
00137
00153           uint64_t
00154           sumDirectoryUsage(const std::string &pathname);
00155
00168           bool fileExists(
00169               const std::string &pathname);
00170
00171           /*
00172            * Indicate whether a path points to a directory.
00173            *
00174            * @param[in] pathname
00175            *  The path to be checked
00176            * @return
00177            *  true if the path is a dir, false otherwise.
00178            * @throw Error::StrategyError
00179            *      An error occurred when using the
00180            *      underlying storage system, or the
00181            *      name is malformed.
00182            */
00183           bool pathIsDirectory(
00184               const std::string &pathname);
00185
00202           int makePath(const std::string &path, const mode_t mode);
00203
00223           Memory::uint8Array
00224           readFile(
00225               const std::string &path,
00226               std::ios_base::openmode mode = std::ios_base::binary);
00227
00254           void
00255           writeFile(
00256               const uint8_t *data,
00257               const size_t size,
00258               const std::string &path,
00259               std::ios_base::openmode mode = std::ios_base::binary);
00260
00285               void
00286           writeFile(
00287               const Memory::uint8Array data,
00288               const std::string &path,
00289               std::ios_base::openmode mode = std::ios_base::binary);
00290
00311               void
00312           readPipe(
00313               void *data,
00314               size_t size,
00315               int pipeFD);
00316
00334           void
00335           readPipe(
00336                   Memory::uint8Array &data,
00337               int pipeFD);
00338
00359               void
00360           writePipe(
00361               const void *data,
00362               size_t size,
00363               int pipeFD);
00364
00383               void
00384           writePipe(
00385               const Memory::uint8Array &data,
00386               int pipeFD);
00387
00407           bool
00408           isReadable(
00409               const std::string &pathname);
00410
00430           bool
00431           isWritable(
00432               const std::string &pathname);
00433
00463           std::string
00464           createTemporaryFile(
```

```
00465                 const std::string &prefix = "",
00466                 const std::string &parentDir = "/tmp");
00467
00500             FILE*
00501             createTemporaryFile(
00502                 std::string &path,
00503                 const std::string &prefix = "",
00504                 const std::string &parentDir = "/tmp");
00505
00520             uint64_t
00521             countLines(
00522                 const std::string &path);
00523
00535             uint64_t
00536             countLines(
00537                 const Memory::uint8Array &textBuffer);
00538         }
00539     }
00540 }
00541 #endif  /* __BE_IO_UTILITY_H */
```

I.65 be_iris.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IRIS_H__
00012 #define __BE_IRIS_H__
00013
00014 #include <be_framework_enumeration.h>
00015
00016 namespace BiometricEvaluation
00017 {
00029     namespace Iris
00030     {
00035         enum class CaptureDeviceTechnology {
00036             Unknown = 0,
00037             CMOSCCD = 1
00038         };
00039
00044         enum class EyeLabel {
00045             Undefined = 0,
00046             Right = 1,
00047             Left = 2
00048         };
00049
00054         enum class ImageType {
00055             Uncropped = 1,
00056             VGA = 2,
00057             Cropped = 3,
00058             CroppedMasked = 7
00059         };
00060
00065         enum class Orientation {
00066             Undefined = 0,
00067             Base = 1,
00068             Flipped = 2
00069         };
00070
00075         enum class ImageCompression {
00076             Undefined = 0,
00077             LosslessNone = 1,
00078             Lossy = 2
00079         };
00080
00085         enum class CameraRange {
00086             Unassigned = 0,
00087             Failed = 1,
```

```
00088             Overflow = 2
00089         };
00090     }
00091 }
00092
00093 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00094     BiometricEvaluation::Iris::CaptureDeviceTechnology,
00095     BE_Iris_CaptureDeviceTechnology_EnumToStringMap);
00096
00097 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00098     BiometricEvaluation::Iris::EyeLabel,
00099     BE_Iris_EyeLabel_EnumToStringMap);
00100
00101 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00102     BiometricEvaluation::Iris::ImageType,
00103     BE_Iris_ImageType_EnumToStringMap);
00104
00105 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00106     BiometricEvaluation::Iris::Orientation,
00107     BE_Iris_Orientation_EnumToStringMap);
00108
00109 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00110     BiometricEvaluation::Iris::ImageCompression,
00111     BE_Iris_ImageCompression_EnumToStringMap);
00112
00113 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00114     BiometricEvaluation::Iris::CameraRange,
00115     BE_Iris_CameraRange_EnumToStringMap);
00116
00117 #endif /* __BE_IRIS_H__ */
00118
```

I.66 be_iris_incitsview.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IRIS_INCITSVIEW_H__
00012 #define __BE_IRIS_INCITSVIEW_H__
00013
00014 #include <string>
00015 #include <vector>
00016
00017 #include <be_iris.h>
00018 #include <be_image.h>
00019 #include <be_memory_indexedbuffer.h>
00020 #include <be_view_view.h>
00021
00022 namespace BiometricEvaluation
00023 {
00024     namespace Iris
00025     {
00038         class INCITSView : public View::View {
00039         public:
00044             typedef struct {
00045                 uint8_t     score;
00046                 uint16_t    vendorID;
00047                 uint16_t    algorithmID;
00048             } QualitySubBlock;
00049             typedef std::vector<QualitySubBlock> QualitySet;
00050
00051             static const uint16_t RANGE_UNASSIGNED = 0;
00052             static const uint16_t RANGE_FAILED = 1;
00053             static const uint16_t RANGE_OVERFLOW = 65535;
00054
00055             static const uint16_t ROLL_ANGLE_UNDEF = 65535;
00056             static const uint16_t ROLL_UNCERTAIN_UNDEF = 65535;
00057             static const uint16_t COORDINATE_UNDEF = 0;
```

```
00058
00065             uint8_t getCertificationFlag() const;
00066
00073             std::string getCaptureDateString() const;
00074
00081             Iris::CaptureDeviceTechnology
00082                 getCaptureDeviceTechnology() const;
00083
00090             uint16_t getCaptureDeviceVendor() const;
00091
00098             uint16_t getCaptureDeviceType() const;
00099
00106             void getQualitySet(
00107                 Iris::INCITSView::QualitySet &qualitySet) const;
00108
00115             Iris::EyeLabel getEyeLabel() const;
00116
00123             Iris::ImageType getImageType() const;
00124
00135             void getImageProperties(
00136                 BiometricEvaluation::Iris::Orientation
00137                 &horizontalOrientation,
00138                 BiometricEvaluation::Iris::Orientation
00139                 &verticalOrientation,
00140                 BiometricEvaluation::Iris::ImageCompression
00141                 &compressionHistory
00142             ) const;
00143
00153             uint16_t getCameraRange();
00154
00163             void getRollAngleInfo(
00164                 uint16_t &rollAngle,
00165                 uint16_t &rollAngleUncertainty);
00166
00185             void getIrisCenterInfo(
00186                 uint16_t &irisCenterSmallestX,
00187                 uint16_t &irisCenterSmallestY,
00188                 uint16_t &irisCenterLargestX,
00189                 uint16_t &irisCenterLargestY,
00190                 uint16_t &irisDiameterSmallest,
00191                 uint16_t &irisDiameterLargest
00192             );
00193
00194         protected:
00195
00196             static const uint32_t ISO2011_STANDARD = 1;
00197             static const uint32_t BASE_FORMAT_ID = 0x49495200;
00198             /* 'I''I''R' 'nul' */
00199             static const uint8_t CAPTURE_DATE_LENGTH = 9;
00200
00201             INCITSView();
00202
00222             INCITSView(
00223                 const std::string &filename,
00224                 const uint32_t viewNumber);
00225
00243             INCITSView(
00244                 const Memory::uint8Array &buffer,
00245                 const uint32_t viewNumber);
00246
00254             Memory::uint8Array const& getIIRData() const;
00255
00275             virtual void readHeader(
00276                 BiometricEvaluation::Memory::IndexedBuffer &buf,
00277                 const uint32_t formatStandard);
00278
00293             virtual void readIrisView(
00294                 Memory::IndexedBuffer &buf);
00295
00296         private:
00297             BiometricEvaluation::Memory::uint8Array _iir;
00298             uint8_t _certFlag;
00299
00300             BiometricEvaluation::Iris::CaptureDeviceTechnology
00301                 _captureDeviceTechnology;
00302
00303             BiometricEvaluation::Iris::INCITSView::QualitySet
```

```
00304                 _qualitySet;
00305
00306             BiometricEvaluation::Iris::EyeLabel
00307                 _eyeLabel;
00308             BiometricEvaluation::Iris::ImageType
00309                 _imageType;
00310             Iris::Orientation _horizontalOrientation;
00311             Iris::Orientation _verticalOrientation;
00312             Iris::ImageCompression _compressionHistory;
00313
00314             uint16_t _cameraRange;
00315             uint16_t _rollAngle;
00316             uint16_t _rollAngleUncertainty;
00317
00318             uint16_t _irisCenterSmallestX;
00319             uint16_t _irisCenterSmallestY;
00320             uint16_t _irisCenterLargestX;
00321             uint16_t _irisCenterLargestY;
00322             uint16_t _irisDiameterSmallest;
00323             uint16_t _irisDiameterLargest;
00324
00325             uint16_t _captureDeviceVendor;
00326             uint16_t _captureDeviceType;
00327             uint8_t _captureDate[CAPTURE_DATE_LENGTH];
00328             std::string _captureDateString;
00329         };
00330     }
00331 }
00332 #endif /* __BE_IRIS_INCITSVIEW_H__ */
00333
```

I.67 be_iris_iso2011view.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_IRIS_ISO2011VIEW_H__
00012 #define __BE_IRIS_ISO2011VIEW_H__
00013
00014 #include <be_iris_incitsview.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Iris
00019     {
00028         class ISO2011View : public Iris::INCITSView {
00029         public:
00034             ISO2011View();
00035
00050             ISO2011View(
00051                 const std::string &filename,
00052                 const uint32_t viewNumber);
00053
00067             ISO2011View(
00068                 const Memory::uint8Array &buffer,
00069                 const uint32_t viewNumber);
00070
00071         protected:
00072             static const uint32_t BASE_SPEC_VERSION = 0x30323000;
00073             /* '0''2''0' 'nul' */
00074
00075             void readISOHeader(
00076                 BiometricEvaluation::Memory::IndexedBuffer &buf);
00077         private:
00078         };
00079     }
00080 }
00081 #endif /* __BE_IRIS_ISO2011VIEW_H__ */
```

```
00082
```

I.68 be_latent_an2kview.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_LATENT_AN2KVIEW_H__
00012 #define __BE_LATENT_AN2KVIEW_H__
00013
00014 #include <be_view_an2kview_varres.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Latent
00019     {
00020         class AN2KView : public View::AN2KViewVariableResolution {
00021         public:
00022
00030             AN2KView(
00031                 const std::string &filename,
00032                 const uint32_t recordNumber);
00033
00042             AN2KView(
00043                 Memory::uint8Array &buf,
00044                 const uint32_t recordNumber);
00045
00056             Feature::FGPSet
00057             getPositions() const;
00058
00067             QualityMetricSet
00068             getLatentQualityMetric()
00069                 const;
00070
00075             Finger::PositionDescriptors
00076             getSearchPositionDescriptors()
00077                 const;
00078
00086             PrintPositionCoordinateSet
00087             getPrintPositionCoordinates()
00088                 const;
00089
00090         protected:
00091         private:
00092         };
00093     }
00094 }
00095 #endif /* __BE_LATENT_AN2KVIEW_H__ */
00096
```

I.69 be_memory.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_MEMORY__
00012 #define __BE_MEMORY__
00013
```

```
00014 #include <memory>
00015 #include <type_traits>
00016
00017 namespace BiometricEvaluation {
00025     namespace Memory
00026     {
00035         template<class T>
00036         struct unique_if
00037         {
00039             using unique_single = std::unique_ptr<T>;
00040         };
00041
00051         template<class T>
00052         struct unique_if<T[]>
00053         {
00055             using unique_array_unknown_bound = std::unique_ptr<T[]>;
00056         };
00057
00066         template<class T, size_t S>
00067         struct unique_if<T[S]>
00068         {
00070             using unique_array_known_bound = void;
00071         };
00072
00086         template<typename T, typename... Ts>
00087         typename unique_if<T>::unique_single
00088         make_unique(Ts&&... params)
00089         {
00090             return (std::unique_ptr<T>
00091                 (new T(std::forward<Ts>(params)...)));
00092         }
00093
00107         template<class T>
00108         typename unique_if<T>::unique_array_unknown_bound
00109         make_unique(size_t size)
00110         {
00111             typedef typename std::remove_extent<T>::type U;
00112             return (std::unique_ptr<T>(new U[size]()));
00113         }
00114
00127         template<class T, class... Ts>
00128         typename unique_if<T>::unique_array_known_bound
00129         make_unique(Ts&&...) = delete;
00130
00138         inline bool
00139         isLittleEndian()
00140         {
00141             /* Anonymous union */
00142             union { uint32_t i; uint8_t c; } u;
00143             u.i = 1;
00144
00145             /*
00146              * Explore contents of octet 1 via properties of union.
00147              *
00148              *      i = 0x00000001
00149              *
00150              *      |1  |2  |3  |4
00151              *      |-------|-------|-------|-------
00152              * Little Endian|01 |00 |00 |00
00153              *    Big Endian|00 |00 |00 |01
00154              */
00155             return (u.c == 1);
00156         }
00157     }
00158 }
00159 #endif /* __BE_MEMORY__ */
00160
```

I.70 be_memory_autoarray.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
```

```
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 /*
00012  * Adapted from "c_array" from "The C++ Programming Language" by Bjarne
00013  * Stroustrup (ISBN: 0201700735).
00014  */
00015
00016 #ifndef __BE_MEMORY_AUTOARRAY_H__
00017 #define __BE_MEMORY_AUTOARRAY_H__
00018
00019 #include <algorithm>
00020 #include <cstddef>
00021 #include <cstdint>
00022 #include <cstring>
00023 #include <limits>
00024 #include <stdexcept>
00025 #include <utility>
00026 #include <vector>
00027
00028 #include <be_error_exception.h>
00029 #include <be_memory_autoarrayiterator.h>
00030
00031 namespace BiometricEvaluation
00032 {
00033     namespace Memory
00034     {
00044         template<class T>
00045         class AutoArray
00046         {
00047             public:
00049                 using value_type = T;
00051                 using size_type = size_t;
00052
00054                 using iterator = AutoArrayIterator<false, T>;
00056                 using const_iterator =
00057                     AutoArrayIterator<true, T>;
00058
00060                 using reference = T&;
00062                 using const_reference = const T&;
00063
00072                 operator T*();
00073
00082                 operator const T*()
00083                     const;
00084
00097                 reference
00098                 operator[](
00099                     ptrdiff_t index);
00100
00113                 const_reference
00114                 operator[](
00115                     ptrdiff_t index)
00116                     const;
00117
00134                 reference
00135                 at(
00136                     ptrdiff_t index);
00137
00154                 const_reference
00155                 at(
00156                     ptrdiff_t index) const;
00157
00167                 iterator
00168                 begin();
00169
00179                 const_iterator
00180                 begin()
00181                     const;
00182
00192                 const_iterator
00193                 cbegin()
00194                     const;
00195
00205                 iterator
```

```
00206                 end();
00207
00217                 const_iterator
00218                 end()
00219                     const;
00220
00230                 const_iterator
00231                 cend()
00232                     const;
00233
00247                 size_type
00248                 size()
00249                     const;
00250
00266                 void
00267                 resize(
00268                     size_type new_size,
00269                     bool free = false);
00270
00288                 void
00289                 copy(
00290                     const T *buffer);
00291
00308                 void
00309                 copy(
00310                     const T *buffer,
00311                     size_type size);
00312
00330                 std::vector<T>
00331                 to_vector()
00332                     const;
00333
00345                 explicit AutoArray(
00346                     size_type size = 0);
00347
00359                 AutoArray(
00360                     const AutoArray &copy);
00361
00371                 AutoArray(
00372                     AutoArray &&rvalue)
00373                     noexcept;
00374
00382                 AutoArray(
00383                     std::initializer_list<T> ilist);
00384
00400                 AutoArray&
00401                 operator=(
00402                     const AutoArray &other);
00403
00416                 AutoArray&
00417                 operator=(
00418                     AutoArray &&other)
00419                     noexcept(
00420                     noexcept(
00421                     std::swap(std::declval<value_type&>(),
00422                     std::declval<value_type&>()))
00423                     &&
00424                     noexcept(
00425                     std::swap(std::declval<size_type&>(),
00426                     std::declval<size_type&>())));
00427
00428
00430                 ~AutoArray();
00431
00432             private:
00434                 value_type *_data;
00436                 size_type _size;
00438                 size_type _capacity;
00439         };
00440
00441         /*************************************************************/
00442         /* Useful type definitions of an AutoArray of basic types.   */
00443         /*************************************************************/
00444         using uint8Array = AutoArray<uint8_t>;
00445         using uint16Array = AutoArray<uint16_t>;
00446         using uint32Array = AutoArray<uint32_t>;
00447
```

```
00449         template<typename T>
00450         bool
00451         operator==(
00452             const AutoArray<T> &lhs,
00453             const AutoArray<T> &rhs);
00454 
00456         template<typename T>
00457         bool
00458         operator!=(
00459             const AutoArray<T> &lhs,
00460             const AutoArray<T> &rhs);
00461 
00463         template<typename T>
00464         bool
00465         operator<(
00466             const AutoArray<T> &lhs,
00467             const AutoArray<T> &rhs);
00468 
00470         template<typename T>
00471         bool
00472         operator<=(
00473             const AutoArray<T> &lhs,
00474             const AutoArray<T> &rhs);
00475 
00477         template<typename T>
00478         bool
00479         operator>(
00480             const AutoArray<T> &lhs,
00481             const AutoArray<T> &rhs);
00482 
00484         template<typename T>
00485         bool
00486         operator>=(
00487             const AutoArray<T> &lhs,
00488             const AutoArray<T> &rhs);
00489     }
00490 }
00491 
00492 /*****************************************************************************/
00493 /* Method implementations.                                                   */
00494 /*****************************************************************************/
00495 template<class T>
00496 typename BiometricEvaluation::Memory::AutoArray<T>::size_type
00497 BiometricEvaluation::Memory::AutoArray<T>::size()
00498     const
00499 {
00500     return (_size);
00501 }
00502 
00503 template<class T>
00504 void
00505 BiometricEvaluation::Memory::AutoArray<T>::resize(
00506     size_type new_size,
00507     bool free)
00508 {
00509     /* If we've already allocated at least new_size space, then bail */
00510     if (!free && (new_size <= _capacity)) {
00511         _size = new_size;
00512         return;
00513     }
00514 
00515     T* new_data = nullptr;
00516     if (new_size != 0) {
00517         new_data = new (std::nothrow) T[new_size];
00518         if (new_data == nullptr)
00519             throw Error::MemoryError("Could not allocate data");
00520     }
00521 
00522     /* Copy as much data as will fit into the new buffer */
00523     std::copy(&_data[0], &_data[((new_size < _size) ? new_size : _size)],
00524         new_data);
00525 
00526     /* Delete the old buffer and assign the new buffer to this object */
00527     if (_data != nullptr)
00528         delete [] _data;
00529     _data = new_data;
00530     _size = _capacity = new_size;
```

```
00531 }
00532
00533 template<class T>
00534 void
00535 BiometricEvaluation::Memory::AutoArray<T>::copy(
00536     const T *buffer)
00537 {
00538     std::copy(&buffer[0], &buffer[_size], _data);
00539 }
00540
00541 template<class T>
00542 void
00543 BiometricEvaluation::Memory::AutoArray<T>::copy(
00544     const T *buffer,
00545     size_type size)
00546 {
00547     this->resize(size);
00548     std::copy(&buffer[0], &buffer[size], _data);
00549 }
00550
00551 template<class T>
00552 typename BiometricEvaluation::Memory::AutoArray<T>::const_reference
00553 BiometricEvaluation::Memory::AutoArray<T>::at(
00554     ptrdiff_t index) const
00555 {
00556     if (index < 0)
00557         throw std::out_of_range("index");
00558     if ((size_type)index < _size)
00559         return (_data[index]);
00560
00561     throw std::out_of_range("index");
00562 }
00563
00564 template<class T>
00565 typename BiometricEvaluation::Memory::AutoArray<T>::reference
00566 BiometricEvaluation::Memory::AutoArray<T>::at(
00567     ptrdiff_t index)
00568 {
00569     if (index < 0)
00570         throw std::out_of_range("index");
00571     if ((size_type)index < _size)
00572         return (_data[index]);
00573
00574     throw std::out_of_range("index");
00575 }
00576
00577 template<class T>
00578 typename std::vector<T>
00579 BiometricEvaluation::Memory::AutoArray<T>::to_vector()
00580     const
00581 {
00582     return {this->cbegin(), this->cend()};
00583 }
00584
00585 /*****************************************************************************/
00586 /* Conversion Operators.                                                     */
00587 /*****************************************************************************/
00588 template<class T>
00589 BiometricEvaluation::Memory::AutoArray<T>::operator T*()
00590 {
00591     return (_data);
00592 }
00593
00594 template<class T>
00595 BiometricEvaluation::Memory::AutoArray<T>::operator const T*()
00596     const
00597 {
00598     return (_data);
00599 }
00600
00601 /*****************************************************************************/
00602 /* Operator Overloads.                                                       */
00603 /*****************************************************************************/
00604 template<class T>
00605 typename BiometricEvaluation::Memory::AutoArray<T>::reference
00606 BiometricEvaluation::Memory::AutoArray<T>::operator[](
00607     ptrdiff_t index)
```

```
00608 {
00609     return (_data[index]);
00610 }
00611
00612 template<class T>
00613 typename BiometricEvaluation::Memory::AutoArray<T>::const_reference
00614 BiometricEvaluation::Memory::AutoArray<T>::operator[](
00615     ptrdiff_t index)
00616     const
00617 {
00618     return (_data[index]);
00619 }
00620
00621 template<class T>
00622 BiometricEvaluation::Memory::AutoArray<T>&
00623 BiometricEvaluation::Memory::AutoArray<T>::operator=(
00624     const BiometricEvaluation::Memory::AutoArray<T> &other)
00625 {
00626     if (this != &other) {
00627         _size = _capacity = other._size;
00628         if (_data != nullptr) {
00629             delete [] _data;
00630             _data = nullptr;
00631         }
00632         if (_size != 0) {
00633             _data = new (std::nothrow) T[_size];
00634             if (_data == nullptr)
00635                 throw Error::MemoryError("Could not "
00636                     "allocate data");
00637             std::copy(&(other._data[0]), &(other._data[_size]),
00638                 _data);
00639         }
00640     }
00641
00642     return (*this);
00643 }
00644
00645 template<class T>
00646 BiometricEvaluation::Memory::AutoArray<T>&
00647 BiometricEvaluation::Memory::AutoArray<T>::operator=(
00648     BiometricEvaluation::Memory::AutoArray<T> &&other)
00649     noexcept(
00650     noexcept(
00651     std::swap(std::declval<value_type&>(), std::declval<value_type&>())) &&
00652     noexcept(
00653     std::swap(std::declval<size_type&>(), std::declval<size_type&>())))
00654 {
00655     using std::swap;
00656
00657     swap(_size, other._size);
00658     swap(_capacity, other._capacity);
00659     swap(_data, other._data);
00660
00661     return (*this);
00662 }
00663
00664 /*****************************************************************************/
00665 /* Iterators.                                                                */
00666 /*****************************************************************************/
00667 template<class T>
00668 typename BiometricEvaluation::Memory::AutoArray<T>::iterator
00669 BiometricEvaluation::Memory::AutoArray<T>::begin()
00670 {
00671     return (iterator(this, 0));
00672 }
00673
00674 template<class T>
00675 typename BiometricEvaluation::Memory::AutoArray<T>::const_iterator
00676 BiometricEvaluation::Memory::AutoArray<T>::begin()
00677     const
00678 {
00679     return (this->cbegin());
00680 }
00681
00682 template<class T>
00683 typename BiometricEvaluation::Memory::AutoArray<T>::const_iterator
00684 BiometricEvaluation::Memory::AutoArray<T>::cbegin()
```

```
00685     const
00686 {
00687     return (const_iterator(this, 0));
00688 }
00689
00690 template<class T>
00691 typename BiometricEvaluation::Memory::AutoArray<T>::iterator
00692 BiometricEvaluation::Memory::AutoArray<T>::end()
00693 {
00694     if (this->size() > std::numeric_limits<typename BiometricEvaluation::
00695         Memory::AutoArrayIterator<false, T>::difference_type>::max())
00696         throw BiometricEvaluation::Error::StrategyError{"AutoArray too "
00697             "large to represent end iterator"};
00698
00699     return (iterator(this,
00700         static_cast<typename BiometricEvaluation::Memory::
00701         AutoArrayIterator<false, T>::difference_type>(this->size())));
00702 }
00703
00704 template<class T>
00705 typename BiometricEvaluation::Memory::AutoArray<T>::const_iterator
00706 BiometricEvaluation::Memory::AutoArray<T>::end()
00707     const
00708 {
00709     return (this->cend());
00710 }
00711
00712 template<class T>
00713 typename BiometricEvaluation::Memory::AutoArray<T>::const_iterator
00714 BiometricEvaluation::Memory::AutoArray<T>::cend()
00715     const
00716 {
00717     if (this->size() > std::numeric_limits<typename BiometricEvaluation::
00718         Memory::AutoArrayIterator<true, T>::difference_type>::max())
00719         throw BiometricEvaluation::Error::StrategyError{"AutoArray too "
00720             "large to represent end iterator"};
00721
00722     return (const_iterator(this,
00723         static_cast<typename BiometricEvaluation::Memory::
00724         AutoArrayIterator<true, T>::difference_type>(this->size())));
00725 }
00726
00727 /*****************************************************************************/
00728 /* Constructors.                                                             */
00729 /*****************************************************************************/
00730 template<class T>
00731 BiometricEvaluation::Memory::AutoArray<T>::AutoArray(
00732     size_type size) :
00733     _data(nullptr),
00734     _size(size),
00735     _capacity(size)
00736 {
00737     if (_size != 0) {
00738         _data = new (std::nothrow) T[_size];
00739         if (_data == nullptr)
00740             throw Error::MemoryError("Could not allocate data");
00741     }
00742 }
00743
00744 template<class T>
00745 BiometricEvaluation::Memory::AutoArray<T>::AutoArray(
00746     const AutoArray& copy) :
00747     _data(nullptr),
00748     _size(copy._size),
00749     _capacity(copy._size)
00750 {
00751     if (_size != 0) {
00752         _data = new (std::nothrow) T[_size];
00753         if (_data == nullptr)
00754             throw Error::MemoryError("Could not allocate data");
00755         std::copy(&(copy._data[0]), &(copy._data[_size]), _data);
00756     }
00757 }
00758
00759 template<class T>
00760 BiometricEvaluation::Memory::AutoArray<T>::AutoArray(
00761     AutoArray &&rvalue)
```

```
00762     noexcept :
00763     _data(rvalue._data),
00764     _size(rvalue._size),
00765     _capacity(rvalue._capacity)
00766 {
00767     /* Modify for a speedy destruction */
00768     rvalue._data = nullptr;
00769     rvalue._capacity = 0;
00770     rvalue._size = 0;
00771 }
00772
00773 template<class T>
00774 BiometricEvaluation::Memory::AutoArray<T>::AutoArray(
00775     std::initializer_list<T> ilist) : AutoArray(ilist.size())
00776 {
00777     std::copy(ilist.begin(), ilist.end(), _data);
00778 }
00779
00780 /*****************************************************************************/
00781 /* Destructor.                                                               */
00782 /*****************************************************************************/
00783 template<class T>
00784 BiometricEvaluation::Memory::AutoArray<T>::~AutoArray()
00785 {
00786     if (_data != nullptr)
00787         delete [] _data;
00788 }
00789
00790 /*****************************************************************************/
00791 /* Comparison operators.                                                     */
00792 /*****************************************************************************/
00793
00794 template<typename T>
00795 bool
00796 BiometricEvaluation::Memory::operator==(
00797    const typename BiometricEvaluation::Memory::AutoArray<T> &lhs,
00798    const typename BiometricEvaluation::Memory::AutoArray<T> &rhs)
00799 {
00800     if (lhs.size() != rhs.size())
00801         return (false);
00802
00803     return (std::equal(lhs.cbegin(), lhs.cend(), rhs.cbegin()));
00804 }
00805
00806 template<typename T>
00807 bool
00808 BiometricEvaluation::Memory::operator!=(
00809    const typename BiometricEvaluation::Memory::AutoArray<T> &lhs,
00810    const typename BiometricEvaluation::Memory::AutoArray<T> &rhs)
00811 {
00812     return (!(lhs == rhs));
00813 }
00814
00815 template<typename T>
00816 bool
00817 BiometricEvaluation::Memory::operator<(
00818    const typename BiometricEvaluation::Memory::AutoArray<T> &lhs,
00819    const typename BiometricEvaluation::Memory::AutoArray<T> &rhs)
00820 {
00821     return (std::lexicographical_compare(lhs.cbegin(), lhs.cend(),
00822         rhs.cbegin(), rhs.cend()));
00823 }
00824
00825 template<typename T>
00826 bool
00827 BiometricEvaluation::Memory::operator<=(
00828    const typename BiometricEvaluation::Memory::AutoArray<T> &lhs,
00829    const typename BiometricEvaluation::Memory::AutoArray<T> &rhs)
00830 {
00831     return (!(rhs < lhs));
00832 }
00833
00834 template<typename T>
00835 bool
00836 BiometricEvaluation::Memory::operator>(
00837    const typename BiometricEvaluation::Memory::AutoArray<T> &lhs,
00838    const typename BiometricEvaluation::Memory::AutoArray<T> &rhs)
```

```
00839 {
00840     return (rhs < lhs);
00841 }
00842
00843 template<typename T>
00844 bool
00845 BiometricEvaluation::Memory::operator>=(
00846    const typename BiometricEvaluation::Memory::AutoArray<T> &lhs,
00847    const typename BiometricEvaluation::Memory::AutoArray<T> &rhs)
00848 {
00849     return (!(lhs < rhs));
00850 }
00851
00852 #endif /* __BE_MEMORY_AUTOARRAY_H__ */
00853
```

I.71 be_memory_autoarrayiterator.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to Title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_MEMORY_AUTOARRAYITERATOR_H__
00012 #define __BE_MEMORY_AUTOARRAYITERATOR_H__
00013
00014 #include <iterator>
00015 #include <type_traits>
00016
00017 namespace BiometricEvaluation
00018 {
00019     namespace Memory
00020     {
00022         template <typename T> class AutoArray;
00023
00034         template <bool C, class T>
00035         class AutoArrayIterator
00036         {
00037         public:
00038             /*
00039              * Satisfy std::iterator_traits<> expectations.
00040              */
00041
00043             using iterator_category =
00044                 std::random_access_iterator_tag;
00046             using value_type = typename
00047                 std::conditional<C, const T, T>::type;
00049             using difference_type = std::ptrdiff_t;
00051             using pointer = typename
00052                 std::conditional<C, const T*, T*>::type;
00054             using reference = typename
00055                 std::conditional<C, const T&, T&>::type;
00056
00062             using container = typename std::conditional<C,
00063                 const AutoArray<T>*, AutoArray<T>*>::type;
00064
00065             /*
00066              * Constructors
00067              */
00068
00079             AutoArrayIterator(
00080                 container autoArray = nullptr,
00081                 difference_type offset = 0) :
00082                 _autoArray(autoArray),
00083                 _offset(offset)
00084             {
00085
00086             }
00087
00089             AutoArrayIterator(
```

```
00090                 const AutoArrayIterator &rhs) = default;
00092             AutoArrayIterator(
00093                 AutoArrayIterator &&rhs) = default;
00095             ~AutoArrayIterator() = default;
00096
00097             /*
00098              * Assignments
00099              */
00100
00102             inline AutoArrayIterator&
00103             operator=(
00104                 pointer rhs)
00105             {
00106                 _offset = rhs;
00107                 return (*this);
00108             }
00109
00111             inline AutoArrayIterator&
00112             operator=(
00113                 const AutoArrayIterator &rhs) = default;
00114
00116             inline AutoArrayIterator&
00117             operator+=(
00118                 const difference_type &rhs)
00119             {
00120                 _offset += rhs;
00121                 return (*this);
00122             }
00123
00125             inline AutoArrayIterator&
00126             operator-=(
00127                 const difference_type &rhs)
00128             {
00129                 _offset -= rhs;
00130                 return (*this);
00131             }
00132
00133             /*
00134              * Dereferencing Content
00135              */
00136
00138             inline reference
00139             operator*()
00140                 const
00141             {
00142                 return (_autoArray->operator[](_offset));
00143             }
00144
00146             inline pointer
00147             operator->()
00148                 const
00149             {
00150                 return (&(_autoArray->operator[](_offset)));
00151             }
00152
00154             inline reference
00155             operator[](
00156                 const difference_type &rhs)
00157                 const
00158             {
00159                 return (_autoArray->operator[](rhs));
00160             }
00161
00162             /*
00163              * Arithmetic
00164              */
00165
00167             inline AutoArrayIterator&
00168             operator++()
00169             {
00170                 ++_offset;
00171                 return (*this);
00172             }
00173
00175             inline AutoArrayIterator&
00176             operator--()
00177             {
```

```
00178                 --_offset;
00179                 return (*this);
00180             }
00181
00183             inline AutoArrayIterator
00184             operator++(
00185                 int)
00186             {
00187                 const AutoArrayIterator previous(*this);
00188                 ++(*this);
00189                 return (previous);
00190             }
00191
00193             inline AutoArrayIterator
00194             operator--(
00195                 int)
00196             {
00197                 AutoArrayIterator previous(*this);
00198                 --(*this);
00199                 return (previous);
00200             }
00201
00206             inline AutoArrayIterator
00207             operator+(
00208                 const AutoArrayIterator &rhs)
00209                 const
00210             {
00211                 return (AutoArrayIterator(_offset +
00212                     rhs._offset));
00213             }
00214
00216             inline difference_type
00217             operator-(
00218                 const AutoArrayIterator<C, T> &rhs)
00219                 const
00220             {
00221                 return (_offset - rhs._offset);
00222             }
00223
00225             inline AutoArrayIterator
00226             operator+(
00227                 const difference_type &rhs)
00228                 const
00229             {
00230                 return (AutoArrayIterator(_autoArray,
00231                     _offset + rhs));
00232             }
00233
00235             inline AutoArrayIterator
00236             operator-(
00237                 const difference_type &rhs)
00238                 const
00239             {
00240                 return (AutoArrayIterator(_autoArray,
00241                     _offset - rhs));
00242             }
00243
00245             friend inline AutoArrayIterator
00246             operator+(
00247                 const difference_type &lhs,
00248                 const AutoArrayIterator &rhs)
00249             {
00250                 return (AutoArrayIterator(rhs._autoArray,
00251                     lhs + rhs._offset));
00252             }
00253
00258             friend inline AutoArrayIterator
00259             operator-(
00260                 const difference_type &lhs,
00261                 const AutoArrayIterator &rhs)
00262             {
00263                 return (AutoArrayIterator(rhs._autoArray,
00264                     lhs - rhs._offset));
00265             }
00266
00267             /*
00268              * Comparisons
```

```
00269             */
00270
00272             inline bool
00273             operator==(
00274                 const AutoArrayIterator &rhs)
00275                 const
00276             {
00277                 return (_offset == rhs._offset);
00278             }
00279
00281             inline bool
00282             operator!=(
00283                 const AutoArrayIterator &rhs)
00284                 const
00285             {
00286                 return (_offset != rhs._offset);
00287             }
00288
00290             inline bool
00291             operator>(
00292                 const AutoArrayIterator &rhs)
00293                 const
00294             {
00295                 return (_offset > rhs._offset);
00296             }
00297
00299             inline bool
00300             operator<(
00301                 const AutoArrayIterator &rhs)
00302                 const
00303             {
00304                 return (_offset < rhs._offset);
00305             }
00306
00308             inline bool
00309             operator>=(
00310                 const AutoArrayIterator &rhs)
00311                 const
00312             {
00313                 return (_offset >= rhs._offset);
00314             }
00315
00317             inline bool
00318             operator<=(
00319                 const AutoArrayIterator &rhs)
00320                 const
00321             {
00322                 return (_offset <= rhs._offset);
00323             }
00324
00325         private:
00327             container _autoArray;
00329             difference_type _offset;
00330         };
00331     }
00332 }
00333
00334 #endif /* __BE_MEMORY_AUTOARRAYITERATOR_H__ */
```

I.72 be_memory_autoarrayutility.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to Title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_MEMORY_AUTOARRAYUTILITY_H__
00012 #define __BE_MEMORY_AUTOARRAYUTILITY_H__
00013
00014 #include <cstdarg>
```

```
00015 #include <cstdio>
00016 #include <string>
00017 #include <type_traits>
00018
00019 #include <be_error_exception.h>
00020 #include <be_memory_autoarray.h>
00021
00022 namespace BiometricEvaluation
00023 {
00024     namespace Memory
00025     {
00027         namespace AutoArrayUtility
00028         {
00039             template <typename T, typename = typename
00040                 std::enable_if<std::is_same<T, uint8_t>::value ||
00041                 std::is_same<T, char>::value>::type>
00042             inline char *
00043             cstr(
00044                 const AutoArray<T> &rahc)
00045             {
00046                 return ((char *)&(*rahc));
00047             }
00048
00064             template <typename T, typename = typename
00065                 std::enable_if<std::is_same<T, uint8_t>::value ||
00066                 std::is_same<T, char>::value>::type>
00067             inline std::string
00068             getString(
00069                 const AutoArray<T> &aa,
00070                 typename AutoArray<T>::size_type count)
00071             {
00072                 if (count > aa.size())
00073                     throw Error::MemoryError();
00074
00075                 return (std::string(cstr(aa), count));
00076             }
00077
00087             template <typename T, typename = typename
00088                 std::enable_if<std::is_same<T, uint8_t>::value ||
00089                 std::is_same<T, char>::value>::type>
00090             inline void
00091             setString(
00092                 AutoArray<T> &aa,
00093                 const std::string &str)
00094             {
00095                 aa.resize(str.size() + 1);
00096                 ::snprintf(cstr(aa), aa.size(), "%s",
00097                     str.c_str());
00098             }
00099
00111             template <typename T, typename = typename
00112                 std::enable_if<std::is_same<T, uint8_t>::value ||
00113                 std::is_same<T, char>::value>::type>
00114             inline void
00115             setString(
00116                 AutoArray<T> &aa,
00117                 const char *str,
00118                 ...)
00119             {
00120                 aa.resize(strlen(str) + 1);
00121
00122                 va_list args;
00123                 va_start(args, str);
00124                 ::vsnprintf(cstr(aa), aa.size(), str, args);
00125                 va_end(args);
00126             }
00127         }
00128     }
00129 }
00130
00141 template <typename T, typename = typename
00142     std::enable_if<std::is_same<T, uint8_t>::value ||
00143     std::is_same<T, char>::value>::type>
00144 inline std::string
00145 to_string(
00146     const BiometricEvaluation::Memory::AutoArray<T> &aa)
00147 {
```

```
00148     return (std::string(
00149         BiometricEvaluation::Memory::AutoArrayUtility::cstr(aa),
00150         aa.size() - 1));
00151 }
00152
00153 #endif /* __BE_MEMORY_AUTOARRAYUTILITY_H__ */
```

I.73 be_memory_autobuffer.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_MEMORY_AUTOBUFFER__
00012 #define __BE_MEMORY_AUTOBUFFER__
00013
00014 #include <be_error_exception.h>
00015
00016 namespace BiometricEvaluation {
00017
00018     namespace Memory {
00019     template<class T>
00020     class AutoBuffer {
00021
00054         public:
00055
00056             using value_type = T;
00057
00058             using reference = T&;
00059             using const_reference = const T&;
00060
00061             operator T*();
00062             T* operator->();
00063
00064             AutoBuffer& operator= (const AutoBuffer& other);
00065
00066             AutoBuffer();
00067             /* You already have an allocated buffer */
00068             AutoBuffer(T* data);
00069             /* Constructor, Destructor, Copy Copy Constructor */
00070             AutoBuffer(int (*ctor)(T**), void (*dtor)(T*),
00071                 int (*copyCtor)(T**, T*)=nullptr);
00072             AutoBuffer(const AutoBuffer& copy);
00073
00074             ~AutoBuffer();
00075
00076         private:
00077
00078             /* Pointer to allocated data */
00079             T* _data;
00080             /* Allocator function pointer */
00081             int (*_ctor)(T**);
00082             /* Destructor function pointer */
00083             void (*_dtor)(T*);
00084             /* Copy constructor function pointer */
00085             int (*_copyCtor)(T**, T*);
00086             /*
00087              * True if we passed in preallocated data.  Don't
00088              * perform any memory management, just keep track of
00089              * the pointer.
00090              */
00091             bool _handsOff;
00092
00093     };
00094     }
00095 }
00096
00097 /*****************************************************************************/
00098 /* Implementation.                                                           */
```

```
00099 /*****************************************************************************/
00100
00101 /*****************************************************************************/
00102 /* Method implementations.                                                   */
00103 /*****************************************************************************/
00104
00105 /*
00106  * Operators.
00107  */
00108 template<class T>
00109 BiometricEvaluation::Memory::AutoBuffer<T>::operator T*()
00110 {
00111     return _data;
00112 }
00113
00114 template<class T>
00115 T* BiometricEvaluation::Memory::AutoBuffer<T>::operator->()
00116 {
00117     return _data;
00118 }
00119
00120 template<class T>
00121 BiometricEvaluation::Memory::AutoBuffer<T>&
00122 BiometricEvaluation::Memory::AutoBuffer<T>::operator=
00123     (const BiometricEvaluation::Memory::AutoBuffer<T>& copy)
00124 {
00125     if (this != &copy) {
00126         /* Copy function pointers, they aren't changing */
00127         _ctor = copy._ctor;
00128         _dtor = copy._dtor;
00129         _copyCtor = copy._copyCtor;
00130         _handsOff = copy._handsOff;
00131
00132         if (_handsOff)
00133             /* Just copy the pointer */
00134             _data = copy._data;
00135         else {
00136             /*
00137              * Use copy constructor on the allocated memory to
00138              * duplicate.
00139              */
00140             if (_copyCtor == nullptr)
00141                 throw Error::ParameterError("Copy "
00142                     "constructor is nullptr");
00143             if ((_copyCtor)(&_data, copy._data) != 0)
00144                 throw Error::DataError("Data could not be "
00145                     "allocated");
00146         }
00147     }
00148
00149     return *this;
00150 }
00151
00152 /*****************************************************************************/
00153 /* Constructors.                                                             */
00154 /*****************************************************************************/
00155 template<class T>
00156 BiometricEvaluation::Memory::AutoBuffer<T>::AutoBuffer()
00157 {
00158     _data = nullptr;
00159     _handsOff = true;
00160 }
00161
00162 template<class T>
00163 BiometricEvaluation::Memory::AutoBuffer<T>::AutoBuffer(
00164     int (*ctor)(T**),
00165     void (*dtor)(T*),
00166     int (*copyCtor)(T**, T*))
00167 {
00168     if (ctor != nullptr)
00169         _ctor = ctor;
00170     else
00171         throw Error::ParameterError("Allocator is nullptr");
00172
00173     if (dtor != nullptr)
00174         _dtor = dtor;
00175     else
```

```
00176         throw Error::ParameterError("Destructor is nullptr");
00177
00178     /* Don't require copy constructor, user might never make copy */
00179     _copyCtor = copyCtor;
00180
00181     /* Initial allocation the data */
00182     if ((_ctor)(&_data) != 0)
00183         throw Error::DataError("Data could not be allocated");
00184
00185     _handsOff = false;
00186 }
00187
00188 template<class T>
00189 BiometricEvaluation::Memory::AutoBuffer<T>::AutoBuffer(T* data)
00190 {
00191     /*
00192      * With this constructor, the AutoBuffer is essentially nothing more
00193      * than a bloated pointer.  The caller still must free memory manually.
00194      * This just allows for uniform usage in classes that can take an
00195      * allocated buffer or can create one.
00196      */
00197     _data = data;
00198     _handsOff = true;
00199 }
00200
00201 template<class T>
00202 BiometricEvaluation::Memory::AutoBuffer<T>::AutoBuffer(const AutoBuffer& copy)
00203 {
00204     /* Copy function pointers, they aren't changing */
00205     _ctor = copy._ctor;
00206     _dtor = copy._dtor;
00207     _copyCtor = copy._copyCtor;
00208     _handsOff = copy._handsOff;
00209
00210     if (_handsOff)
00211         /* Just copy the pointer and pray the user hasn't freed it */
00212         _data = copy._data;
00213     else {
00214         /* Use copy constructor on the allocated memory to duplicate */
00215         if (_copyCtor == nullptr)
00216             throw Error::ParameterError("Copy constructor is "
00217                 "nullptr");
00218
00219         if ((_copyCtor)(&_data, copy._data) != 0)
00220             throw Error::DataError("Data could not be "
00221                 "allocated");
00222     }
00223 }
00224
00225 /*****************************************************************************/
00226 /* Destructor.                                                               */
00227 /*****************************************************************************/
00228 template<class T>
00229 BiometricEvaluation::Memory::AutoBuffer<T>::~AutoBuffer()
00230 {
00231     if (!_handsOff)
00232         (_dtor)(_data);
00233 }
00234
00235 #endif /* __BE_MEMORY_AUTOBUFFER__ */
```

I.74 be_memory_indexedbuffer.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_MEMORY_INDEXEDBUFFER__
00012 #define __BE_MEMORY_INDEXEDBUFFER__
```

```
00013
00014 #include <be_memory_autoarray.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Memory
00019     {
00033         class IndexedBuffer
00034         {
00035             public:
00037                 IndexedBuffer();
00038
00048                 IndexedBuffer(
00049                     const uint8_t *data,
00050                     uint64_t size);
00051
00059                 IndexedBuffer(
00060                     const uint8Array &aa);
00061
00063                 IndexedBuffer(
00064                     const IndexedBuffer &copy) = default;
00065
00073                 uint32_t
00074                 getSize()
00075                     const;
00076
00088                 uint32_t
00089                 getIndex()
00090                     const;
00091
00102                 void
00103                 setIndex(
00104                     uint64_t index);
00105
00118                 uint8_t
00119                 scanU8Val();
00120
00133                 uint16_t
00134                 scanU16Val();
00135
00149                 uint16_t
00150                 scanBeU16Val();
00151
00165                 uint32_t
00166                 scanU32Val();
00167
00181                 uint32_t
00182                 scanBeU32Val();
00183
00197                 uint64_t
00198                 scanU64Val();
00199
00216                 uint64_t
00217                 scan(
00218                     void *buf,
00219                     uint64_t len);
00220
00228                 virtual const uint8_t*
00229                 get()
00230                     const;
00231
00233 #ifdef __MIC__
00234                 virtual ~IndexedBuffer() noexcept = default;
00235 #else
00236                 virtual ~IndexedBuffer() = default;
00237 #endif
00238
00239
00240             private:
00242                 const uint8_t * const _data;
00243
00245                 const uint64_t _size;
00246
00248                 uint64_t _index;
00249         };
00250     }
00251 }
```

```
00252 #endif /* __BE_MEMORY_INDEXEDBUFFER__ */
```

I.75 be_memory_mutableindexedbuffer.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_MEMORY_MUTABLEINDEXEDBUFFER__
00012 #define __BE_MEMORY_MUTABLEINDEXEDBUFFER__
00013
00014 #include <be_memory_indexedbuffer.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Memory
00019     {
00021         class MutableIndexedBuffer : public IndexedBuffer
00022         {
00023             public:
00033                 MutableIndexedBuffer(
00034                     uint8_t *data,
00035                     uint64_t size);
00036
00044                 MutableIndexedBuffer(
00045                     uint8Array &aa);
00046
00048                 MutableIndexedBuffer(
00049                     const MutableIndexedBuffer &copy) = default;
00050
00068                 uint64_t
00069                 push(
00070                     const void *buf,
00071                     uint64_t len);
00072
00087                 uint8_t
00088                 pushU8Val(
00089                     uint8_t val);
00090
00105                 uint16_t
00106                 pushU16Val(
00107                     uint16_t val);
00108
00124                 uint16_t
00125                 pushBeU16Val(
00126                     uint16_t val);
00127
00142                 uint32_t
00143                 pushU32Val(
00144                     uint32_t val);
00145
00161                 uint32_t
00162                 pushBeU32Val(
00163                     uint32_t val);
00164
00179                 uint64_t
00180                 pushU64Val(
00181                     uint64_t val);
00182
00190                 virtual const uint8_t*
00191                 get()
00192                     const;
00193
00195 #ifdef __MIC__
00196                 virtual ~MutableIndexedBuffer() noexcept =
00197                     default;
00198 #else
00199                 virtual ~MutableIndexedBuffer() = default;
00200 #endif
```

```
00201
00202             private:
00204                 uint8_t *_mutableData;
00205         };
00206     }
00207 }
00208 #endif /* __BE_MEMORY_MUTABLEINDEXEDBUFFER__ */
```

I.76 be_memory_orderedmap.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __ORDERED_MAP_H__
00012 #define __ORDERED_MAP_H__
00013
00014 #include <iterator>
00015 #include <list>
00016 #include <memory>
00017 #include <unordered_map>
00018
00019
00020 namespace BiometricEvaluation
00021 {
00022     namespace Memory
00023     {
00024         /* Forward declarations */
00025         template<class Key, class T> class OrderedMap;
00026         template<class Key, class T> class OrderedMapIterator;
00027         template<class Key, class T> class OrderedMapConstIterator;
00028
00030         template<class Key, class T>
00031         class OrderedMapIterator
00032         {
00033         public:
00034             /*
00035              * Satisfy std::iterator_traits<> expectations.
00036              */
00037
00039             using iterator_category =
00040                 std::bidirectional_iterator_tag;
00042             using value_type = std::pair<Key, T>;
00044             using difference_type = std::ptrdiff_t;
00046             using pointer = value_type*;
00048             using reference = value_type&;
00049
00050             friend class OrderedMap<Key, T>;
00051             friend class OrderedMapConstIterator<Key, T>;
00052
00054             OrderedMapIterator();
00056             ~OrderedMapIterator();
00057
00062             reference
00063             operator*()
00064                 const;
00065
00070             pointer
00071             operator->()
00072                 const;
00073
00075             OrderedMapIterator&
00076             operator++();
00077
00079             OrderedMapIterator
00080             operator++(
00081                 int);
00082
00084             OrderedMapIterator&
```

```
00085             operator--();
00086
00088             OrderedMapIterator
00089             operator--(
00090                 int);
00091
00103             bool
00104             operator==(
00105                 const OrderedMapIterator &rhs)
00106                 const;
00107
00119              bool
00120              operator!=(
00121                  const OrderedMapIterator &rhs)
00122                  const;
00123
00124         private:
00135             OrderedMapIterator(
00136                 const OrderedMap<Key, T> *orderedMap,
00137                 const typename std::list<Key>::iterator listIter);
00138
00140             const OrderedMap<Key, T>  *_orderedMap;
00142             typename std::list<Key>::iterator _listIter;
00144             mutable std::pair<Key, T> _currentPair;
00145         };
00146
00148         template<class Key, class T>
00149         class OrderedMapConstIterator
00150         {
00151         public:
00152             /*
00153              * Satisfy std::iterator_traits<> expectations.
00154              */
00155
00157             using iterator_category =
00158                 std::bidirectional_iterator_tag;
00160             using value_type = std::pair<Key, T>;
00162             using difference_type = std::ptrdiff_t;
00164             using pointer = const value_type*;
00166             using reference = const value_type&;
00167
00168             friend class OrderedMap<Key, T>;
00169
00171             OrderedMapConstIterator();
00173             OrderedMapConstIterator(
00174                 const OrderedMapIterator<Key, T> &iterator);
00176             ~OrderedMapConstIterator();
00177
00182             reference
00183             operator*()
00184                 const;
00185
00190             pointer
00191             operator->()
00192                 const;
00193
00195             OrderedMapConstIterator&
00196             operator++();
00197
00199             OrderedMapConstIterator
00200             operator++(
00201                 int);
00202
00204             OrderedMapConstIterator&
00205             operator--();
00206
00208             OrderedMapConstIterator
00209             operator--(
00210                 int);
00211
00223             bool
00224             operator==(
00225                 const OrderedMapConstIterator &rhs)
00226                 const;
00227
00239              bool
00240              operator!=(
```

```
00241                 const OrderedMapConstIterator &rhs)
00242                 const;
00243
00244         private:
00255             OrderedMapConstIterator(
00256                 const OrderedMap<Key, T> *orderedMap,
00257                 const typename std::list<Key>::iterator listIter);
00258
00260             const OrderedMap<Key, T>  *_orderedMap;
00262             mutable typename std::list<Key>::iterator _listIter;
00264             mutable std::pair<Key, T> _currentPair;
00265         };
00266
00267
00272         template<class Key, class T>
00273         class OrderedMap
00274         {
00275         public:
00276             using container = typename std::unordered_map<Key, T>;
00277             using iterator = OrderedMapIterator<Key, T>;
00278             using const_iterator = OrderedMapConstIterator<Key, T>;
00279
00280             using size_type = typename container::size_type;
00281             using value_type = typename container::value_type;
00282             using key_type = Key;
00283             using mapped_type = T;
00284
00285             using key_equal = typename container::key_equal;
00286
00287             friend class OrderedMapIterator<Key, T>;
00288             friend class OrderedMapConstIterator<Key, T>;
00289
00291             OrderedMap();
00292
00307             bool
00308             push_back(
00309                 const value_type &value);
00310
00323             void
00324             erase(
00325                 iterator pos);
00326
00334             void
00335             erase(
00336                 const Key &key);
00337
00342             iterator
00343             begin();
00344
00349             const_iterator
00350             begin()
00351                 const;
00352
00357             const_iterator
00358             cbegin()
00359                 const;
00360
00366             iterator
00367             end();
00368
00374             const_iterator
00375             end()
00376                 const;
00377
00383             const_iterator
00384             cend()
00385                 const;
00386
00391             size_type
00392             size()
00393                 const;
00394
00408             bool
00409             keyExists(
00410                 const Key &key)
00411                 const;
00412
```

```
00420             const OrderedMapIterator<Key, T>
00421             find(
00422                 const Key &key)
00423                 const;
00424
00425             std::shared_ptr<value_type>
00426             find_quick(
00427                 const Key &key)
00428                 const;
00429
00440             T&
00441             operator[](
00442                 const Key &key);
00443
00445             key_equal
00446             key_eq()
00447                 const;
00448
00450             ~OrderedMap();
00451
00452         private:
00454             container *_elements;
00456             std::list<Key> *_ordering;
00457         };
00458     }
00459 }
00460
00461 template<class Key, class T>
00462 BiometricEvaluation::Memory::OrderedMap<Key, T>::OrderedMap() :
00463     _elements(new container()),
00464     _ordering(new std::list<Key>())
00465 {
00466
00467 }
00468
00469 template<class Key, class T>
00470 BiometricEvaluation::Memory::OrderedMap<Key, T>::~OrderedMap()
00471 {
00472     if (_elements != nullptr)
00473         delete _elements;
00474     if (_ordering != nullptr)
00475         delete _ordering;
00476 }
00477
00478 template<class Key, class T>
00479 bool
00480 BiometricEvaluation::Memory::OrderedMap<Key, T>::push_back(
00481     const value_type &value)
00482 {
00483     if (_elements->insert(value).second) {
00484         _ordering->push_back(value.first);
00485         return (true);
00486     } else
00487         return (false);
00488 }
00489
00490 template<class Key, class T>
00491 void
00492 BiometricEvaluation::Memory::OrderedMap<Key, T>::erase(
00493     iterator pos)
00494 {
00495     _ordering->remove(pos->first);
00496     _elements->erase(pos);
00497 }
00498
00499 template<class Key, class T>
00500 void
00501 BiometricEvaluation::Memory::OrderedMap<Key, T>::erase(
00502     const Key &key)
00503 {
00504     _ordering->remove(key);
00505     _elements->erase(_elements->find(key));
00506 }
00507
00508 template<class Key, class T>
00509 typename BiometricEvaluation::Memory::OrderedMap<Key, T>::iterator
00510 BiometricEvaluation::Memory::OrderedMap<Key, T>::begin()
```

```
00511 {
00512     return (OrderedMapIterator<Key, T>(this, _ordering->begin()));
00513 }
00514
00515 template<class Key, class T>
00516 typename BiometricEvaluation::Memory::OrderedMap<Key, T>::const_iterator
00517 BiometricEvaluation::Memory::OrderedMap<Key, T>::begin()
00518     const
00519 {
00520     return (OrderedMapIterator<Key, T>(this, _ordering->begin()));
00521 }
00522
00523 template<class Key, class T>
00524 typename BiometricEvaluation::Memory::OrderedMap<Key, T>::const_iterator
00525 BiometricEvaluation::Memory::OrderedMap<Key, T>::cbegin()
00526     const
00527 {
00528     return (OrderedMapIterator<Key, T>(this, _ordering->begin()));
00529 }
00530
00531 template<class Key, class T>
00532 typename BiometricEvaluation::Memory::OrderedMap<Key, T>::iterator
00533 BiometricEvaluation::Memory::OrderedMap<Key, T>::end()
00534 {
00535     return (OrderedMapIterator<Key, T>(this, _ordering->end()));
00536 }
00537
00538 template<class Key, class T>
00539 typename BiometricEvaluation::Memory::OrderedMap<Key, T>::const_iterator
00540 BiometricEvaluation::Memory::OrderedMap<Key, T>::end()
00541     const
00542 {
00543     return (OrderedMapIterator<Key, T>(this, _ordering->end()));
00544 }
00545
00546 template<class Key, class T>
00547 typename BiometricEvaluation::Memory::OrderedMap<Key, T>::const_iterator
00548 BiometricEvaluation::Memory::OrderedMap<Key, T>::cend()
00549     const
00550 {
00551     return (OrderedMapIterator<Key, T>(this, _ordering->end()));
00552 }
00553
00554 template<class Key, class T>
00555 typename BiometricEvaluation::Memory::OrderedMap<Key, T>::size_type
00556 BiometricEvaluation::Memory::OrderedMap<Key, T>::size()
00557     const
00558 {
00559     return (_elements->size());
00560 }
00561
00562 template<class Key, class T>
00563 bool
00564 BiometricEvaluation::Memory::OrderedMap<Key, T>::keyExists(
00565     const Key &key)
00566     const
00567 {
00568     return (_elements->find(key) != _elements->end());
00569 }
00570
00571 template<class Key, class T>
00572 T&
00573 BiometricEvaluation::Memory::OrderedMap<Key, T>::operator[](
00574     const Key &key)
00575 {
00576     std::pair<typename container::iterator, bool> result =
00577         _elements->insert(std::make_pair(key, T()));
00578
00579     if (result.second) {
00580         /* New insertion */
00581         _ordering->push_back(key);
00582         return (result.first->second);
00583     } else
00584         /* Already in list */
00585         return (result.first->second);
00586 }
00587
```

```
00588 template<class Key, class T>
00589 const BiometricEvaluation::Memory::OrderedMapIterator<Key, T>
00590 BiometricEvaluation::Memory::OrderedMap<Key, T>::find(
00591     const Key &key)
00592     const
00593 {
00594     return (OrderedMapIterator<Key, T>(this,
00595         std::find(_ordering->begin(), _ordering->end(), key)));
00596 }
00597
00598 template<class Key, class T>
00599 std::shared_ptr<
00600 typename BiometricEvaluation::Memory::OrderedMap<Key, T>::value_type>
00601 BiometricEvaluation::Memory::OrderedMap<Key, T>::find_quick(
00602     const Key &key)
00603     const
00604 {
00605     typename container::const_iterator it = _elements->find(key);
00606     if (it != _elements->end())
00607         return (std::shared_ptr<
00608             typename OrderedMap<Key, T>::value_type>(
00609             new typename OrderedMap<Key, T>::value_type(it->first,
00610             it->second)));
00611     return (std::shared_ptr<
00612         typename OrderedMap<Key, T>::value_type>());
00613 }
00614
00615 template<class Key, class T>
00616 typename BiometricEvaluation::Memory::OrderedMap<Key, T>::key_equal
00617 BiometricEvaluation::Memory::OrderedMap<Key, T>::key_eq()
00618     const
00619 {
00620     return (_elements->key_eq());
00621 }
00622
00623 /*
00624  * OrderedMapIterator Implementation
00625  */
00626
00627 template<class Key, class T>
00628 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::OrderedMapIterator() :
00629     _orderedMap(nullptr),
00630     _listIter()
00631 {
00632
00633 }
00634
00635 template<class Key, class T>
00636 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::OrderedMapIterator(
00637     const OrderedMap<Key, T> *orderedMap,
00638     const typename std::list<Key>::iterator listIter) :
00639     _orderedMap(orderedMap),
00640     _listIter(listIter)
00641 {
00642
00643 }
00644
00645 template<class Key, class T>
00646 typename BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::reference
00647 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::operator*()
00648     const
00649 {
00650     _currentPair = *(_orderedMap->_elements->find(*_listIter));
00651     return (_currentPair);
00652 }
00653
00654 template<class Key, class T>
00655 typename BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::pointer
00656 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::operator->()
00657     const
00658 {
00659     _currentPair = *(_orderedMap->_elements->find(*_listIter));
00660     return (&_currentPair);
00661 }
00662
00663 template<class Key, class T>
00664 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>&
```

```
00665 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::operator++()
00666 {
00667     ++_listIter;
00668     return (*this);
00669 }
00670
00671 template<class Key, class T>
00672 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>
00673 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::operator++(
00674     int)
00675 {
00676     OrderedMapIterator previousIterator(*this);
00677     ++(*this);
00678     return (previousIterator);
00679 }
00680
00681 template<class Key, class T>
00682 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>&
00683 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::operator--()
00684 {
00685     --_listIter;
00686     return (*this);
00687 }
00688
00689 template<class Key, class T>
00690 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>
00691 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::operator--(
00692     int)
00693 {
00694     OrderedMapIterator previousIterator(*this);
00695     --(*this);
00696     return (previousIterator);
00697 }
00698
00699 template<class Key, class T>
00700 bool
00701 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::operator==(
00702     const OrderedMapIterator &rhs)
00703     const
00704 {
00705     return ((_orderedMap == rhs._orderedMap) &&
00706         (_listIter == rhs._listIter));
00707 }
00708
00709 template<class Key, class T>
00710 bool
00711 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::operator!=(
00712     const OrderedMapIterator &rhs)
00713     const
00714 {
00715     return (!(this->operator==(rhs)));
00716 }
00717
00718 template<class Key, class T>
00719 BiometricEvaluation::Memory::OrderedMapIterator<Key, T>::~OrderedMapIterator()
00720 {
00721     /* Don't delete _orderedMap, we don't own it. */
00722 }
00723
00724 /*
00725  * OrderedMapConstIterator Implementation
00726  */
00727
00728 template<class Key, class T>
00729 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::
00730 OrderedMapConstIterator() :
00731     _orderedMap(nullptr),
00732     _listIter()
00733 {
00734
00735 }
00736
00737 template<class Key, class T>
00738 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::
00739 OrderedMapConstIterator(
00740     const OrderedMap<Key, T> *orderedMap,
00741     const typename std::list<Key>::iterator listIter) :
```

```
00742     _orderedMap(orderedMap),
00743     _listIter(listIter)
00744 {
00745
00746 }
00747
00748 template<class Key, class T>
00749 typename
00750 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::reference
00751 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::operator*()
00752     const
00753 {
00754     _currentPair = *(_orderedMap->_elements->find(*_listIter));
00755     return (_currentPair);
00756 }
00757
00758 template<class Key, class T>
00759 typename
00760 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::pointer
00761 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::operator->()
00762     const
00763 {
00764     _currentPair = *(_orderedMap->_elements->find(*_listIter));
00765     return (&_currentPair);
00766 }
00767
00768 template<class Key, class T>
00769 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>&
00770 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::operator++()
00771 {
00772     ++_listIter;
00773     return (*this);
00774 }
00775
00776 template<class Key, class T>
00777 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>
00778 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::operator++(
00779     int)
00780 {
00781     OrderedMapConstIterator previousIterator(*this);
00782     ++(*this);
00783     return (previousIterator);
00784 }
00785
00786 template<class Key, class T>
00787 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>&
00788 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::operator--()
00789 {
00790     --_listIter;
00791     return (*this);
00792 }
00793
00794 template<class Key, class T>
00795 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>
00796 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::operator--(
00797     int)
00798 {
00799     OrderedMapConstIterator previousIterator(*this);
00800     --(*this);
00801     return (previousIterator);
00802 }
00803
00804 template<class Key, class T>
00805 bool
00806 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::operator==(
00807     const OrderedMapConstIterator &rhs)
00808     const
00809 {
00810     return ((_orderedMap == rhs._orderedMap) &&
00811         (_listIter == rhs._listIter));
00812 }
00813
00814 template<class Key, class T>
00815 bool
00816 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::operator!=(
00817     const OrderedMapConstIterator &rhs)
00818     const
```

```
00819 {
00820     return (!(this->operator==(rhs)));
00821 }
00822
00823 template<class Key, class T>
00824 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::
00825 OrderedMapConstIterator(
00826     const OrderedMapIterator<Key, T> &iterator) :
00827     _orderedMap(iterator._orderedMap),
00828     _listIter(iterator._listIter),
00829     _currentPair(iterator._currentPair)
00830 {
00831
00832 }
00833
00834 template<class Key, class T>
00835 BiometricEvaluation::Memory::OrderedMapConstIterator<Key, T>::
00836 ~OrderedMapConstIterator()
00837 {
00838     /* Don't delete _orderedMap, we don't own it. */
00839 }
00840
00841 #endif /* __ORDERED_MAP_H__ */
```

I.77 be_mpi.h

```
00001
00010 #ifndef _BE_MPI_H
00011 #define _BE_MPI_H
00012
00013 #include <memory>
00014 #include <string>
00015
00016 #include <be_framework_enumeration.h>
00017 #include <be_io_logsheet.h>
00018
00019 namespace BiometricEvaluation {
00024     namespace MPI {
00025
00036         std::string generateUniqueID();
00037
00044         void printStatus(const std::string &message);
00045
00059         void logEntry(
00060             IO::Logsheet &logsheet);
00061
00074         void logMessage(
00075             IO::Logsheet &logsheet,
00076             const std::string &message);
00077
00097         std::shared_ptr<BiometricEvaluation::IO::Logsheet> openLogsheet(
00098             const std::string &url,
00099             const std::string &description);
00100
00102         enum class TaskCommand : int32_t
00103         {
00105             Continue = 0,
00107             Ignore = 1,
00109             Exit = 2,
00111             QuickExit = 3,
00113             TermExit = 4
00114         };
00115
00117         using taskcmd_t = std::underlying_type<TaskCommand>::type;
00118
00120         enum class TaskStatus : int32_t
00121         {
00123             OK = 0,
00125             Failed = 1,
00127             Exit = 2,
00129             RequestJobTermination = 3
00130         };
00131
00133         using taskstat_t = std::underlying_type<TaskStatus>::type;
00134
```

```
00136         enum class MessageTag : int32_t
00137         {
00139             Control = 0,
00141             Data = 1,
00148             OOB = 2
00149         };
00150
00152         using msgtag_t = std::underlying_type<MessageTag>::type;
00153     }
00154 }
00155
00156 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00157     BiometricEvaluation::MPI::TaskCommand,
00158     BE_MPI_TaskCommand_EnumToStringMap);
00159
00160 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00161     BiometricEvaluation::MPI::TaskStatus,
00162     BE_MPI_TaskStatus_EnumToStringMap);
00163
00164 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00165     BiometricEvaluation::MPI::MessageTag,
00166     BE_MPI_MessageTag_EnumToStringMap);
00167
00168 #endif /* _BE_MPI_H */
00169
```

I.78 be_mpi_csvdistributor.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties.  Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain.  NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_MPI_CSVDISTRIBUTOR_H_
00012 #define BE_MPI_CSVDISTRIBUTOR_H_
00013
00014 #include <memory>
00015 #include <string>
00016
00017 #include <be_mpi_csvresources.h>
00018 #include <be_mpi_distributor.h>
00019
00020 namespace BiometricEvaluation
00021 {
00022     namespace MPI
00023     {
00042         class CSVDistributor : public Distributor
00043         {
00044         public:
00049             static const std::string CHECKPOINTLINECOUNT;
00050
00055             static const std::string CHECKPOINTRANDOMSEED;
00056
00066             CSVDistributor(
00067                 const std::string &propertiesFileName,
00068                 const std::string &delimiter = "");
00069             ~CSVDistributor();
00070
00071         protected:
00072             void
00073             createWorkPackage(MPI::WorkPackage &workPackage);
00074             void
00075             checkpointSave(const std::string &reason);
00076             void
00077             checkpointRestore();
00078
00079         private:
00080             std::unique_ptr<MPI::CSVResources> _resources;
00081             uint64_t _distributedLineCount{};
00082         };
```

```
00083     }
00084 }
00085
00086 #endif /* BE_MPI_CSVDISTRIBUTOR_H_ */
00087
```

I.79 be_mpi_csvprocessor.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties.  Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain.  NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_MPI_CSVPROCESSOR_H_
00012 #define BE_MPI_CSVPROCESSOR_H_
00013
00014 #include <be_mpi_csvresources.h>
00015 #include <be_mpi_workpackageprocessor.h>
00016
00017 namespace BiometricEvaluation
00018 {
00019     namespace MPI
00020     {
00030         class CSVProcessor : public WorkPackageProcessor
00031         {
00032         public:
00058             CSVProcessor(
00059                 const std::string &propertiesFileName);
00060
00061             virtual ~CSVProcessor() = default;
00062
00082             virtual void
00083             processLine(
00084                 const uint64_t lineNum,
00085                 const std::string &line) = 0;
00086
00087 #if 0
00104             virtual void
00105             processTokens(
00106                 const uint64_t lineNum,
00107                 const std::vector<std::string> &tokens) = 0;
00108 #endif
00109
00110             /* Implement WorkPackageProcessor interface */
00111             virtual std::shared_ptr<WorkPackageProcessor>
00112             newProcessor(
00113                 std::shared_ptr<IO::Logsheet> &logsheet) = 0;
00114
00115             virtual void
00116             performInitialization(
00117                 std::shared_ptr<IO::Logsheet> &logsheet) = 0;
00118
00119             void processWorkPackage(
00120                 MPI::WorkPackage &workPackage);
00121
00122         protected:
00123             std::shared_ptr<MPI::CSVResources>
00124             getResources();
00125
00126         private:
00127             std::shared_ptr<MPI::CSVResources> _resources;
00128         };
00129     }
00130 }
00131
00132 #endif /* BE_MPI_CSVPROCESSOR_H_ */
00133
```

I.80 be_mpi_csvresources.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties.  Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain.  NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_MPI_CSVRESOURCES_H_
00012 #define BE_MPI_CSVRESOURCES_H_
00013
00014 #include <random>
00015 #include <string>
00016 #include <vector>
00017 #include <utility>
00018
00019 #include <be_memory_autoarray.h>
00020 #include <be_mpi_resources.h>
00021
00022 namespace BiometricEvaluation
00023 {
00024     namespace MPI
00025     {
00026         class CSVResources : public Resources
00027         {
00028         public:
00030             static const std::string INPUTCSVPROPERTY;
00032             static const std::string CHUNKSIZEPROPERTY;
00034             static const std::string USEBUFFERPROPERTY;
00036             static const std::string RANDOMIZEPROPERTY;
00038             static const std::string RANDOMSEEDPROPERTY;
00040             static const std::string DELIMITERPROPERTY;
00042             static const std::string TRIMPROPERTY;
00043
00044             static std::vector<std::string>
00045             getRequiredProperties();
00046
00047             static std::vector<std::string>
00048             getOptionalProperties();
00049
00050             CSVResources(
00051                 const std::string &propertiesFileName);
00052             ~CSVResources();
00053
00054             uint32_t
00055             getChunkSize()
00056                 const;
00057
00068             bool
00069             useBuffer()
00070                 const;
00071
00081             bool
00082             randomizeLines()
00083                 const;
00084
00093             uint64_t
00094             getNumRemainingLines()
00095                 const;
00096
00098             std::string
00099             getDelimiter()
00100                 const;
00101
00119             std::pair<uint64_t, std::string>
00120             readLine();
00121
00132             uint64_t
00133             getNumLines()
00134                 const;
00135
00146             std::mt19937_64::result_type
00147             getRandomSeed()
```

```
00148                 const;
00149
00150         private:
00164             void
00165             openCSV();
00166
00167             std::pair<uint64_t, std::string>
00168             readLine(
00169                 bool randomize);
00170
00171             uint32_t _chunkSize;
00172
00174             uint64_t _numLines;
00176             uint64_t _remainingLines;
00177
00179             std::string _csvPath;
00181             std::shared_ptr<std::ifstream> _csvStream;
00182
00184             bool _trimWhitespace;
00186             bool _useBuffer;
00188             Memory::uint8Array _csvBuffer;
00190             bool _randomizeLines;
00192             std::vector<std::pair<uint64_t, std::string>>
00193                 _randomizedLines;
00195             std::mt19937_64 _rng;
00197             std::mt19937_64::result_type _rngSeed;
00199             uint64_t _offset;
00200
00202             std::string _delimiter;
00203         };
00204     }
00205 }
00206
00207 #endif /* BE_MPI_CSVRESOURCES_H_ */
00208
```

I.81 be_mpi_distributor.h

```
00001
00010 #ifndef _BE_MPI_DISTRIBUTOR_H
00011 #define _BE_MPI_DISTRIBUTOR_H
00012
00013 #include <memory>
00014 #include <set>
00015 #include <string>
00016
00017 #include <be_error_exception.h>
00018 #include <be_io_logsheet.h>
00019 #include <be_io_propertiesfile.h>
00020 #include <be_mpi.h>
00021 #include <be_mpi_resources.h>
00022 #include <be_mpi_workpackage.h>
00023
00024 namespace BiometricEvaluation {
00025     namespace MPI {
00051         class Distributor {
00052         public:
00057             static const std::string CHECKPOINTFILENAME;
00058
00063             static const std::string CHECKPOINTREASON;
00064
00069             static const std::string CHECKPOINTPID;
00070
00081             Distributor(const std::string &propertiesFileName);
00082
00083             virtual ~Distributor();
00084
00093             void start();
00094
00095         protected:
00104             virtual void createWorkPackage(
00105                 MPI::WorkPackage &workPackage) = 0;
00106
00121             virtual void checkpointSave(
00122                 const std::string &reason) = 0;
```

```
00123
00134             virtual void checkpointRestore() = 0;
00135
00142             std::shared_ptr<IO::Logsheet> getLogsheet() const;
00143
00150             std::shared_ptr<IO::PropertiesFile>
00151             getCheckpointData() const;
00152
00153         private:
00161             void distributeWork();
00162
00167             void sendWorkPackage(
00168                 MPI::WorkPackage &workPackage,
00169                 int MPITask);
00170
00178             void shutdown();
00179
00180             std::unique_ptr<MPI::Resources> _resources;
00181
00182             /* The list of tasks accepting work */
00183             std::set<int> _activeMpiTasks;
00184
00185             std::shared_ptr<IO::Logsheet> _logsheet;
00186             std::shared_ptr<IO::PropertiesFile> _checkpointData;
00187         };
00188     }
00189 }
00190
00191 #endif /* _BE_MPI_DISTRIBUTOR_H */
00192
```

I.82 be_mpi_exception.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties.  Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain.  NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_MPI_EXCEPTION_H_
00012 #define BE_MPI_EXCEPTION_H_
00013
00014 #include <string>
00015
00016 #include <be_error_exception.h>
00017
00018 namespace BiometricEvaluation
00019 {
00020     namespace MPI
00021     {
00022         class Exception : public Error::Exception
00023         {
00024         public:
00026             Exception();
00027
00036             Exception(std::string info);
00037
00039             virtual ~Exception() noexcept = default;
00040         };
00041
00048         class TerminateJob : public Exception
00049         {
00050         public:
00052             TerminateJob();
00053
00062             TerminateJob(std::string info);
00063         };
00064     }
00065 }
00066
00067
```

```
00068 #endif /* BE_MPI_EXCEPTION_H_ */
00069
```

I.83 be_mpi_receiver.h

```
00001
00010 #ifndef _BE_MPI_RECEIVER_H
00011 #define _BE_MPI_RECEIVER_H
00012
00013 #include <string>
00014 #include <vector>
00015 #include <memory>
00016
00017 #include <be_error_exception.h>
00018 #include <be_mpi.h>
00019 #include <be_mpi_resources.h>
00020 #include <be_mpi_workpackage.h>
00021 #include <be_mpi_workpackageprocessor.h>
00022 #include <be_process_forkmanager.h>
00023
00024 namespace BiometricEvaluation {
00025     namespace MPI {
00055         class Receiver {
00056         public:
00069             Receiver(
00070                 const std::string &propertiesFileName,
00071                 const std::shared_ptr<
00072                     BiometricEvaluation::MPI::WorkPackageProcessor>
00073                 &workPackageProcessor);
00074
00075             ~Receiver();
00076
00089             void start();
00090
00091         protected:
00092
00093         private:
00094             MPI::TaskStatus requestWorkPackages();
00095             void sendWorkPackage(MPI::WorkPackage &workPackage);
00096             void startWorkers();
00097             void shutdown(
00098                 const MPI::TaskStatus &status,
00099                 const std::string &reason);
00100
00101             Process::ForkManager _processManager;
00102
00103             std::shared_ptr<MPI::WorkPackageProcessor>
00104                 _workPackageProcessor;
00105
00106             std::shared_ptr<MPI::Resources> _resources;
00107             std::shared_ptr<IO::Logsheet> _logsheet;
00108
00109             /*
00110              * Declare the class that implements process worker.
00111              */
00112             class PackageWorker : public Process::Worker
00113             {
00114             public:
00115                 PackageWorker(
00116                 const std::shared_ptr<MPI::WorkPackageProcessor>
00117                     &workPackageProcessor,
00118                 const std::shared_ptr<MPI::Resources>
00119                     &resources);
00120
00121                 int32_t workerMain();
00122
00123                 ~PackageWorker();
00124
00125             private:
00126                 std::shared_ptr<
00127                 BiometricEvaluation::MPI::WorkPackageProcessor>
00128                     _workPackageProcessor;
00129                 std::shared_ptr<MPI::Resources> _resources;
00130                 std::shared_ptr<IO::Logsheet> _logsheet;
00131             };
```

```
00132         };
00133     }
00134 }
00135
00136 #endif /* _BE_MPI_RECEIVER_H */
00137
```

I.84 be_mpi_recordprocessor.h

```
00001
00010 #ifndef _BE_MPI_RECORDPROCESSOR_H
00011 #define _BE_MPI_RECORDPROCESSOR_H
00012
00013 #include <be_mpi_recordstoreresources.h>
00014 #include <be_mpi_workpackageprocessor.h>
00015
00016 namespace BiometricEvaluation {
00017     namespace MPI {
00027         class RecordProcessor : public WorkPackageProcessor {
00028         public:
00054             RecordProcessor(
00055                 const std::string &propertiesFileName);
00056
00057             virtual ~RecordProcessor();
00058
00076             //XXX the thrown exception should be refined into
00077             //XXX particular classes
00078             virtual void processRecord(const std::string &key) = 0;
00079
00096             virtual void processRecord(
00097                 const std::string &key,
00098                 const Memory::uint8Array &value) = 0;
00099
00100             /* Implement WorkPackageProcessor interface */
00101             virtual std::shared_ptr<WorkPackageProcessor>
00102                 newProcessor(
00103                 std::shared_ptr<IO::Logsheet> &logsheet) = 0;
00104
00105             virtual void performInitialization(
00106                 std::shared_ptr<IO::Logsheet> &logsheet) = 0;
00107
00108             void processWorkPackage(
00109                 MPI::WorkPackage &workPackage);
00110
00111         protected:
00112             std::shared_ptr<MPI::RecordStoreResources>
00113                  getResources();
00114         private:
00115             std::shared_ptr<MPI::RecordStoreResources>
00116                  _resources;
00117         };
00118     }
00119 }
00120
00121 #endif /* _BE_MPI_RECORDPROCESSOR_H */
00122
```

I.85 be_mpi_recordstoredistributor.h

```
00001
00010 #ifndef _BE_MPI_RECORDSTOREDISTRIBUTOR_H
00011 #define _BE_MPI_RECORDSTOREDISTRIBUTOR_H
00012
00013 #include <be_mpi_distributor.h>
00014 #include <be_mpi_recordstoreresources.h>
00015
00016 namespace BiometricEvaluation {
00017     namespace MPI {
00018
00030         class RecordStoreDistributor : public Distributor {
00031         public:
00035             static const std::string CHECKPOINTLASTKEY;
```

```
00036
00040             static const std::string CHECKPOINTNUMKEYS;
00041
00073             RecordStoreDistributor(
00074                 const std::string &propertiesFileName,
00075                 const bool includeValues);
00076
00077             ~RecordStoreDistributor();
00078
00079         protected:
00080             void
00081             createWorkPackage(MPI::WorkPackage &workPackage);
00082             void checkpointSave(const std::string &reason);
00083             void checkpointRestore();
00084
00085         private:
00086             std::unique_ptr<MPI::RecordStoreResources>
00087                 _resources;
00088             uint64_t _recordsRemaining;
00089             bool _includeValues;
00090             std::string _lastDistributedKey{};
00091         };
00092     }
00093 }
00094
00095 #endif /* _BE_MPI_RECORDSTOREDISTRIBUTOR_H */
00096
```

I.86 be_mpi_recordstoreresources.h

```
00001
00010 #ifndef _BE_MPI_RECORDSTORERESOURCES_H
00011 #define _BE_MPI_RECORDSTORERESOURCES_H
00012
00013 #include <be_io_recordstore.h>
00014 #include <be_mpi_resources.h>
00015
00016 namespace BiometricEvaluation {
00017     namespace MPI {
00028         class RecordStoreResources : public Resources {
00029         public:
00034             static const std::string INPUTRSPROPERTY;
00039             static const std::string CHUNKSIZEPROPERTY;
00040
00047             static std::vector<std::string> getRequiredProperties();
00048
00055             static std::vector<std::string>
00056                 getOptionalProperties();
00057
00070             RecordStoreResources(
00071                 const std::string &propertiesFileName);
00072
00073             ~RecordStoreResources();
00074
00075             uint32_t getChunkSize() const;
00076
00084             bool haveRecordStore() const;
00085
00091              std::shared_ptr<IO::RecordStore>
00092                 getRecordStore() const;
00093
00094         private:
00095             uint32_t _chunkSize;
00096             std::shared_ptr<IO::RecordStore> _recordStore{};
00097         };
00098     }
00099 }
00100
00101 #endif /* _BE_MPI_RECORDSTORERESOURCES_H */
00102
```

I.87 be_mpi_resources.h

```
00001
00010 #ifndef _BE_MPI_RESOURCES_H
00011 #define _BE_MPI_RESOURCES_H
00012
00013 #include <memory>
00014 #include <string>
00015 #include <vector>
00016
00017 namespace BiometricEvaluation {
00018     namespace MPI {
00025         class Resources {
00026         public:
00034             static const std::string WORKERSPERNODEPROPERTY;
00035
00043             static const std::string NUMCPUS;
00044
00052             static const std::string NUMCORES;
00053
00061             static const std::string NUMSOCKETS;
00062
00067             static const std::string LOGSHEETURLPROPERTY;
00068
00074             static const std::string CHECKPOINTPATHPROPERTY;
00075
00082             static std::vector<std::string>
00083                 getRequiredProperties();
00084
00091             static std::vector<std::string>
00092                 getOptionalProperties();
00093
00107             Resources(const std::string &propertiesFileName);
00108
00116             std::string getPropertiesFileName() const;
00117
00128             std::string getLogsheetURL() const;
00129
00139             std::string getCheckpointPath() const;
00140
00141             ~Resources();
00142
00143             int getRank() const;
00144             int getNumTasks() const;
00145             int getWorkersPerNode() const;
00146
00147         private:
00148             std::string _propertiesFileName;
00149             int _rank;
00150             int _numTasks;
00151             int _workersPerNode;
00152             std::string _logsheetURL;
00153             std::string _checkpointPath;
00154         };
00155     }
00156 }
00157
00158 #endif /* _BE_MPI_RESOURCES_H */
00159
```

I.88 be_mpi_runtime.h

```
00001
00010 #ifndef _BE_MPI_RUNTIME_H
00011 #define _BE_MPI_RUNTIME_H
00012
00013 #include <string>
00014
00015 #include <be_mpi.h>
00016 #include <be_mpi_distributor.h>
00017 #include <be_mpi_receiver.h>
00018
00019 namespace BiometricEvaluation {
00020     namespace MPI {
```

```
00021
00022         extern bool Exit;   /* Exit signal was received */
00023         extern bool QuickExit;  /* Quick exit signal received */
00024         extern bool TermExit;   /* Immediate exit signal received */
00025
00026         extern bool checkpointEnable;
00027         extern bool doCheckpointRestore;
00028
00039         class Runtime {
00040         public:
00061             Runtime(
00062                 int &argc,
00063                 char** &argv,
00064                 bool checkpointEnable = false);
00065
00066             ~Runtime();
00067
00081             void start(
00082                 BiometricEvaluation::MPI::Distributor &distributor,
00083                 BiometricEvaluation::MPI::Receiver &receiver);
00084
00092             void shutdown();
00093
00105             void abort(int errcode);
00106
00107         private:
00108             int _argc;
00109             char **_argv;
00110         };
00111     }
00112 }
00113
00114 #endif /* _BE_MPI_RUNTIME_H */
00115
```

I.89 be_mpi_workpackage.h

```
00001
00010 #ifndef _BE_MPI_WORKPACKAGE_H
00011 #define _BE_MPI_WORKPACKAGE_H
00012
00013 #include <be_memory_autoarray.h>
00014
00015 namespace BiometricEvaluation {
00016     namespace MPI {
00025         class WorkPackage {
00026         public:
00031             WorkPackage();
00032
00040             WorkPackage(const Memory::uint8Array &data);
00041             ~WorkPackage();
00042
00047             void getData(Memory::uint8Array &data) const;
00048
00055             void setData(const Memory::uint8Array &data);
00056
00063             uint64_t getSize() const;
00064
00075             uint64_t getNumElements() const;
00076
00084             void setNumElements(const uint64_t numElements);
00085
00086         protected:
00087         private:
00088             Memory::uint8Array _data;
00089             uint64_t _numElements;
00090         };
00091     }
00092 }
00093
00094 #endif /* _BE_MPI_WORKPACKAGE_H */
00095
```

I.90 be_mpi_workpackageprocessor.h

```
00001
00010 #ifndef _BE_MPI_WORKPACKAGEPROCESSOR_H
00011 #define _BE_MPI_WORKPACKAGEPROCESSOR_H
00012
00013 #include <memory>
00014 #include <be_io_logsheet.h>
00015 #include <be_mpi_workpackage.h>
00016
00021 namespace BiometricEvaluation {
00022     namespace MPI {
00023
00044         class WorkPackageProcessor {
00045         public:
00061             virtual std::shared_ptr<WorkPackageProcessor>
00062                 newProcessor(
00063                 std::shared_ptr<IO::Logsheet> &logsheet) = 0;
00064
00085             virtual void performInitialization(
00086                 std::shared_ptr<IO::Logsheet> &logsheet) = 0;
00087
00099             virtual void processWorkPackage(
00100                 MPI::WorkPackage &workPackage) = 0;
00101
00120             virtual void performShutdown();
00121
00129             void
00130             setLogsheet(std::shared_ptr<IO::Logsheet> &logsheet);
00131
00139             std::shared_ptr<IO::Logsheet> getLogsheet();
00140
00141             virtual ~WorkPackageProcessor();
00142
00143         protected:
00144         private:
00145             std::shared_ptr<IO::Logsheet> _logsheet;
00146         };
00147     }
00148 }
00149
00150 #endif /* _BE_MPI_WORKPACKAGEPROCESSOR_H */
00151
```

I.91 be_palm.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PALM_H__
00012 #define __BE_PALM_H__
00013
00014 #include <iostream>
00015 #include <map>
00016 #include <vector>
00017
00018 #include <be_framework_enumeration.h>
00019
00020 namespace BiometricEvaluation
00021 {
00029     namespace Palm
00030     {
00038         enum class Position
00039         {
00040             Unknown         = 20,
00041             RightFull       = 21,
00042             RightWriters        = 22,
```

```
00043             LeftFull        = 23,
00044             LeftWriters     = 24,
00045             RightLower      = 25,
00046             RightUpper      = 26,
00047             LeftLower       = 27,
00048             LeftUpper       = 28,
00049             RightOther      = 29,
00050             LeftOther       = 30,
00051             RightInterdigital   = 31,
00052             RightThenar     = 32,
00053             RightHypothenar     = 33,
00054             LeftInterdigital    = 34,
00055             LeftThenar      = 35,
00056             LeftHypothenar      = 36,
00057             RightGrasp      = 37,
00058             LeftGrasp       = 38,
00059             RightCarpelDelta    = 81,
00060             LeftCarpelDelta     = 82,
00061             RightFullWithWriters    = 83,
00062             LeftFullWithWriters = 84,
00063             RightWristBracelet  = 85,
00064             LeftWristBracelet   = 86
00065         };
00066     }
00067 }
00068 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00069     BiometricEvaluation::Palm::Position,
00070     BE_Palm_Position_EnumToStringMap);
00071
00072 #endif /* __BE_PALM_H__ */
00073
```

I.92 be_palm_an2kview.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PALM_AN2KVIEW_H__
00012 #define __BE_PALM_AN2KVIEW_H__
00013
00014 #include <be_view_an2kview_varres.h>
00015
00016 namespace BiometricEvaluation
00017 {
00018     namespace Palm
00019     {
00030         class AN2KView : public View::AN2KViewVariableResolution {
00031         public:
00032
00040             AN2KView(
00041                 const std::string &filename,
00042                 const uint32_t recordNumber);
00043
00051             AN2KView(
00052                 BiometricEvaluation::Memory::uint8Array &buf,
00053                 const uint32_t recordNumber);
00054
00061             Palm::Position
00062             getPosition() const;
00063
00072             QualityMetricSet
00073             getPalmQualityMetric() const;
00074
00075         protected:
00076         private:
00077             void readImageRecord(const RecordType typeID);
00078         };
00079     }
```

```
00080 }
00081 #endif /* __BE_PALM_AN2KVIEW_H__ */
00082
```

I.93 be_plantar.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PLANTAR_H__
00012 #define __BE_PLANTAR_H__
00013
00014 #include <iostream>
00015 #include <map>
00016 #include <vector>
00017
00018 #include <be_framework_enumeration.h>
00019
00020 namespace BiometricEvaluation
00021 {
00027     namespace Plantar
00028     {
00036         enum class Position
00037         {
00038             UnknownSole     = 60,
00039             RightSole       = 61,
00040             LeftSole        = 62,
00041             UnknownToe      = 63,
00042             RightBigToe     = 64,
00043         };
00044     }
00045 }
00046
00047 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00048     BiometricEvaluation::Plantar::Position,
00049     BE_Plantar_Position_EnumToStringMap);
00050
00051 #endif /* __BE_PLANTAR_H__ */
00052
```

I.94 be_process.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_H__
00012 #define __BE_PROCESS_H__
00013
00014 #include <map>
00015 #include <memory>
00016 #include <string>
00017
00018 namespace BiometricEvaluation
00019 {
00027     namespace Process
00028     {
00030         using ParameterList =
00031             std::map<std::string, std::shared_ptr<void>>;
```

```
00032     }
00033 }
00034 #endif /* __BE_PROCESS_H__ */
```

I.95 be_process_commandcenter.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to Title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_COMMANDCENTER_H__
00012 #define __BE_PROCESS_COMMANDCENTER_H__
00013
00014 #include <cstdio>
00015 #include <string>
00016 #include <vector>
00017
00018 #include <be_framework_enumeration.h>
00019 #include <be_memory_autoarray.h>
00020 #include <be_memory_autoarrayutility.h>
00021 #include <be_process_messagecenter.h>
00022 #include <be_text.h>
00023
00024 namespace BiometricEvaluation
00025 {
00026     namespace Process
00027     {
00029         template <typename T,
00030             typename = typename std::enable_if<std::is_enum<T>::value>>
00031         class CommandCenter
00032         {
00033         public:
00035             class Command
00036             {
00037             public:
00039                 uint32_t clientID;
00041                 T command;
00043                 std::vector<std::string> arguments;
00044             };
00045
00046             static_assert(std::is_enum<T>::value,
00047                 "Invalid templatization of CommandCenter.");
00048
00056              CommandCenter(
00057                 uint16_t port = MessageCenter::DEFAULT_PORT) :
00058                 _messageCenter(port)
00059             {
00060
00061             }
00062
00064             ~CommandCenter() = default;
00065
00079             inline bool
00080             hasPendingCommands()
00081             {
00082                 return (this->_messageCenter.
00083                     hasUnseenMessages());
00084             }
00085
00103             inline bool
00104             getNextCommand(
00105                 Command &command,
00106                 int numSeconds = -1,
00107                 std::string invalidCommandResponse = "")
00108             {
00109                 Memory::uint8Array buffer;
00110                 if (!this->_messageCenter.getNextMessage(
00111                     command.clientID, buffer, numSeconds))
00112                     return (false);
```

```
00113
00114                 /* Arguments are space separated */
00115                 command.arguments = Text::split(
00116                     to_string(buffer), ' ');
00117                 if (command.arguments.size() == 0)
00118                     return (false);
00119
00120                 /* Remove newline from last argument */
00121                 std::for_each(command.arguments.begin(),
00122                     command.arguments.end(),
00123                     [](std::string &i) {
00124                     i = Text::trimWhitespace(i);
00125                 });
00126
00127                 /* Split actual command off of arguments */
00128                 try {
00129                     command.command = BiometricEvaluation::
00130                         Framework::Enumeration::to_enum<T>(
00131                         command.arguments[0]);
00132                 } catch (const Error::ObjectDoesNotExist&) {
00133                     /*
00134                      * Send implementation specific usage
00135                      * if set.
00136                      */
00137                     if (invalidCommandResponse != "")
00138                         this->sendResponse(
00139                             command.clientID,
00140                             invalidCommandResponse);
00141                     else {
00142                         static const std::string
00143                             INVALID = ": "
00144                             "command not recognized";
00145                         this->sendResponse(
00146                             command.clientID,
00147                             command.arguments[0] +
00148                             INVALID);
00149                     }
00150                     return (false);
00151                 }
00152                 command.arguments.erase(
00153                     command.arguments.begin());
00154
00155                 return (true);
00156             }
00157
00171             inline void
00172             sendResponse(
00173                 uint32_t clientID,
00174                 const std::string &response,
00175                 const std::string prefix = ">> ",
00176                 const std::string suffix = "\n")
00177             {
00178                 Memory::uint8Array message;
00179                 Memory::AutoArrayUtility::setString(message,
00180                     prefix + response + suffix);
00181                 this->_messageCenter.sendResponse(clientID,
00182                     message);
00183             }
00184
00192             inline void
00193             disconnectClient(
00194                 uint32_t clientID)
00195             {
00196                 this->sendResponse(clientID, "Goodbye");
00197                 this->_messageCenter.disconnectClient(clientID);
00198             }
00199
00200         private:
00202             MessageCenter _messageCenter;
00203         };
00204
00206         template <typename T>
00207         class CommandParser : public CommandCenter<T>
00208         {
00209         public:
00217             virtual void
00218             parse(
```

```
00219                 const typename
00220                 CommandCenter<T>::Command &command) = 0;
00221
00236             inline bool
00237             getNextCommand(
00238                 typename CommandCenter<T>::Command &command,
00239                 int numSeconds = -1)
00240             {
00241                 return (CommandCenter<T>::getNextCommand(
00242                     command, numSeconds, this->getUsage()));
00243             }
00244
00255             inline void
00256             setUsage(
00257                 const std::string &usage)
00258             {
00259                 this->_usage = usage;
00260             }
00261
00263             inline std::string
00264             getUsage()
00265                 const
00266             {
00267                 return (this->_usage);
00268             }
00269
00277             CommandParser(
00278                 uint16_t port = MessageCenter::DEFAULT_PORT) :
00279                 CommandCenter<T>(port),
00280                 _usage("")
00281             {
00282
00283             }
00284
00286             virtual ~CommandParser() = default;
00287
00288         private:
00290             std::string _usage;
00291         };
00292     }
00293 }
00294
00295 #endif /* __BE_PROCESS_COMMANDCENTER_H__ */
```

I.96 be_process_forkmanager.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_FORKMANAGER_H__
00012 #define __BE_PROCESS_FORKMANAGER_H__
00013
00014 #include <unistd.h>
00015
00016 #include <list>
00017
00018 #include <be_process_manager.h>
00019
00020 namespace BiometricEvaluation
00021 {
00022     namespace Process
00023     {
00024         /* Forward declaration */
00025         class ForkWorkerController;
00026
00032         class ForkManager : public Manager
00033         {
00034         public:
```

```
00046             static std::list<ForkManager*> FORKMANAGERS;
00047
00051             ForkManager();
00052
00063             std::shared_ptr<WorkerController>
00064             addWorker(
00065                 std::shared_ptr<Worker> worker);
00066
00083             void
00084             startWorkers(
00085                 bool wait = true,
00086                 bool communicate = false);
00087
00107             void
00108             startWorker(
00109                 std::shared_ptr<WorkerController> worker,
00110                 bool wait = true,
00111                 bool communicate = false);
00112
00136             void
00137             stopWorker(
00138                 std::shared_ptr<WorkerController> workerController);
00139
00147             void broadcastSignal(int signo);
00148
00161             bool
00162             responsibleFor(
00163                 const pid_t pid)
00164                 const;
00165
00176             void
00177             setNotWorking(
00178                 const pid_t pid);
00179
00185             void
00186             markAllFinished();
00187
00198             bool
00199             getIsWorkingStatus(
00200                 const pid_t pid)
00201                 const;
00202
00203             void
00204             waitForWorkerExit();
00205
00210             ~ForkManager();
00211
00226             void
00227             setExitCallback(
00228                 void (*exitCallback)
00229                 (std::shared_ptr<ForkWorkerController> worker,
00230                 int stat_loc));
00231
00245             static void
00246             defaultExitCallback(
00247                 std::shared_ptr<ForkWorkerController> worker,
00248                 int status);
00249
00266             void
00267             setExitStatus(
00268                 const pid_t pid,
00269                 const int32_t waitStatus);
00270
00271         private:
00286             std::shared_ptr<ForkWorkerController>
00287             getProcessWithPID(
00288                 pid_t pid);
00289
00294             void
00295             _wait();
00296
00305             static void
00306             reap(
00307                 int signal);
00308
00317             class Status {
00318             public:
```

```
00320                Status();
00321
00323                pid_t pid;
00325                bool isWorking;
00326            };
00327
00341            void
00342            (*_exitCallback)
00343                (std::shared_ptr<ForkWorkerController> wc,
00344                int stat_loc);
00345
00347            bool _parent;
00348
00350            std::map<
00351                std::shared_ptr<ForkWorkerController>, Status>
00352                _wcStatus;
00353        };
00354
00355
00360        class ForkWorkerController : public WorkerController
00361        {
00362        public:
00370            bool
00371            isWorking()
00372                const;
00373
00374            bool
00375            everWorked()
00376                const;
00377
00386            void
00387            reset();
00388
00401            pid_t
00402            getPID()
00403                const;
00404
00416            static void
00417            _stop(
00418                int signal);
00419
00424                ~ForkWorkerController();
00425
00426        protected:
00427
00428        private:
00429
00436            ForkWorkerController(
00437                std::shared_ptr<Worker> worker);
00438
00456            void
00457            start(
00458                bool communicate = false);
00459
00469            void
00470            stop();
00471
00473                pid_t _pid;
00474
00480            static std::shared_ptr<Worker> _staticWorker;
00481
00482            /*
00483             * Friends.
00484             */
00485
00502            friend void
00503            ForkManager::startWorkers(
00504                bool wait,
00505                bool communicate);
00506
00526            friend void
00527            ForkManager::startWorker(
00528                std::shared_ptr<WorkerController> worker,
00529                bool wait,
00530                bool communicate);
00531
00548            friend void
```

```
00549             ForkManager::stopWorker(
00550                 std::shared_ptr<WorkerController> workerController);
00551
00562             friend std::shared_ptr<WorkerController>
00563             ForkManager::addWorker(
00564                 std::shared_ptr<Worker> worker);
00565
00582             friend void
00583             ForkManager::setExitStatus(
00584                 const pid_t pid,
00585                 const int32_t waitStatus);
00586         };
00587     }
00588 }
00589
00590 #endif /* __BE_PROCESS_FORKMANAGER_H__ */
```

I.97 be_process_manager.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_MANAGER_H__
00012 #define __BE_PROCESS_MANAGER_H__
00013
00014 #include <vector>
00015
00016 #include <be_error_exception.h>
00017 #include <be_process.h>
00018 #include <be_process_worker.h>
00019 #include <be_process_workercontroller.h>
00020
00021 namespace BiometricEvaluation
00022 {
00023     namespace Process
00024     {
00029         class Manager
00030         {
00031         public:
00036             Manager();
00037
00048             virtual std::shared_ptr<WorkerController>
00049             addWorker(
00050                 std::shared_ptr<Worker> worker) = 0;
00051
00062             virtual uint32_t
00063             getNumCompletedWorkers() const;
00064
00076             virtual uint32_t
00077             getNumActiveWorkers() const;
00078
00087             virtual uint32_t
00088             getTotalWorkers()
00089                 const;
00090
00107             virtual void
00108             startWorkers(
00109                 bool wait = true,
00110                 bool communicate = false) = 0;
00111
00137             virtual void
00138             startWorker(
00139                 std::shared_ptr<WorkerController> worker,
00140                 bool wait = true,
00141                 bool communicate = false) = 0;
00142
00150             virtual void
00151             waitForWorkerExit() = 0;
```

```
00152
00160             virtual void
00161             reset();
00162
00176             virtual void
00177             stopWorker(
00178                 std::shared_ptr<WorkerController> worker) = 0;
00179
00197             virtual bool
00198             waitForMessage(
00199                 std::shared_ptr<WorkerController> &sender,
00200                 int *nextFD = nullptr,
00201                 int numSeconds = -1)
00202                 const;
00203
00225             virtual bool
00226             getNextMessage(
00227                 std::shared_ptr<WorkerController> &sender,
00228                 Memory::uint8Array &message,
00229                 int numSeconds = -1) const;
00230
00241             virtual void
00242             broadcastMessage(
00243                 Memory::uint8Array &message) const;
00244
00249             virtual ~Manager();
00250
00251         protected:
00256             virtual void
00257             _wait() = 0;
00258
00260             std::vector<std::shared_ptr<WorkerController>>
00261                 _workers;
00262
00264             std::vector<std::shared_ptr<WorkerController>>
00265                 _pendingExit;
00266
00267         private:
00268
00269         };
00270     }
00271 }
00272
00273
00274 #endif /* __BE_PROCESS_MANAGER_H__ */
```

I.98 be_process_mclistener.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to Title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_MESSAGECENTERLISTENER__
00012 #define __BE_PROCESS_MESSAGECENTERLISTENER__
00013
00014 #include <cstdint>
00015 #include <memory>
00016
00017 #include <be_process_forkmanager.h>
00018 #include <be_process_worker.h>
00019
00020 namespace BiometricEvaluation
00021 {
00022     namespace Process
00023     {
00025         class MessageCenterListener : public Worker
00026         {
00027         public:
00029             static const std::string PARAM_PORT;
```

```
00030
00031             int32_t
00032             workerMain();
00033
00034             /* Default constructor. */
00035             MessageCenterListener() = default;
00036             /* Default destructor. */
00037             ~MessageCenterListener() = default;
00038
00039         private:
00041             uint16_t _port;
00043             int _socket;
00045             struct addrinfo *_addr;
00047             std::shared_ptr<Process::ForkManager> _manager;
00049             std::map<uint32_t, std::shared_ptr<WorkerController>>
00050                 _clientMap;
00051
00053             void
00054             parseArgs();
00055
00057             void
00058             spawnReceiver(
00059                 int clientSocket);
00060
00062             void
00063             setupSocket();
00064
00066             void
00067             listen();
00068
00070             int
00071             accept();
00072
00074             void
00075             tearDown();
00076         };
00077     }
00078 }
00079
00080 #endif /* __BE_PROCESS_MESSAGECENTERLISTENER__ */
```

I.99 be_process_mcreceiver.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to Title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_MESSAGECENTERRECEIVER__
00012 #define __BE_PROCESS_MESSAGECENTERRECEIVER__
00013
00014 #include <cstdint>
00015 #include <string>
00016
00017 #include <be_process_worker.h>
00018
00019 namespace BiometricEvaluation
00020 {
00021     namespace Process
00022     {
00028         class MessageCenterReceiver : public Worker
00029         {
00030         public:
00032             int32_t
00033             workerMain();
00034
00036             MessageCenterReceiver() = default;
00038             ~MessageCenterReceiver() = default;
00039
00041             static const std::string PARAM_CLIENT_SOCKET;
```

```
00043             static const std::string PARAM_CLIENT_ID;
00045             static const std::string MSG_DISCONNECT;
00046
00047         private:
00049             int _clientSocket;
00051             uint32_t _clientID;
00052
00054             void
00055             parseArgs();
00056
00071             Memory::uint8Array
00072             receive()
00073                 const;
00074
00087             void
00088             send(
00089                 const Memory::uint8Array &message)
00090                 const;
00091         };
00092     }
00093 }
00094
00095 #endif /* __BE_PROCESS_MESSAGECENTERRECEIVER__ */
```

I.100 be_process_mcutility.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to Title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #include <sys/select.h>
00012
00013 #include <cstdint>
00014 #include <memory>
00015
00016 #include <be_memory_autoarray.h>
00017
00018 #ifndef __BE_PROCESS_MESSAGECENTERUTILITY__
00019 #define __BE_PROCESS_MESSAGECENTERUTILITY__
00020
00021 namespace BiometricEvaluation
00022 {
00023     namespace Process
00024     {
00025         namespace MessageCenterUtility
00026         {
00035             fd_set
00036             fileDescriptorSet(
00037                 int fd);
00038
00052             struct timeval *
00053             createTimeout(
00054                 struct timeval &timeout,
00055                 int32_t numSeconds = 0);
00056
00057
00059             enum class DescriptorType
00060             {
00062                 Read,
00064                 Write,
00066                 Error
00067             };
00068
00090             bool
00091             dataAvailable(
00092                 int fd,
00093                 int32_t numSeconds,
00094                 DescriptorType type = DescriptorType::Read);
00095
```

```
00106             uint32_t
00107             getClientID(
00108                 const Memory::uint8Array &message);
00109
00122             Memory::uint8Array
00123             setClientID(
00124                 uint32_t clientID,
00125                 Memory::uint8Array &message);
00126
00139             Memory::uint8Array
00140             setClientID(
00141                 uint32_t clientID,
00142                 const Memory::uint8Array &message);
00143
00154             Memory::uint8Array
00155             getMessage(
00156                 const Memory::uint8Array &message);
00157         }
00158     }
00159 }
00160
00161 #endif /* __BE_PROCESS_MESSAGECENTERUTILITY__ */
```

I.101 be_process_messagecenter.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to Title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_MESSAGECENTER__
00012 #define __BE_PROCESS_MESSAGECENTER__
00013
00014 #include <memory>
00015
00016 #include <be_process_mclistener.h>
00017 #include <be_process_manager.h>
00018 #include <be_process_workercontroller.h>
00019
00020 namespace BiometricEvaluation
00021 {
00022     namespace Process
00023     {
00025         class MessageCenter
00026         {
00027         public:
00029             static const int CONNECTION_BACKLOG = 10;
00031             static const uint16_t DEFAULT_PORT = 7899;
00033             static const int DEFAULT_TIMEOUT = 1;
00035             static const uint64_t MAX_MESSAGE_LENGTH = 255;
00036
00044             MessageCenter(
00045                 uint32_t port = MessageCenter::DEFAULT_PORT);
00046
00057             bool
00058             hasUnseenMessages()
00059                 const;
00060
00076             bool
00077             getNextMessage(
00078                 uint32_t &clientID,
00079                 Memory::uint8Array &message,
00080                 int numSeconds = -1);
00081
00091             void
00092             sendResponse(
00093                 uint32_t clientID,
00094                 const Memory::uint8Array &message)
00095                 const;
00096
```

```
00104             void
00105             disconnectClient(
00106                 uint32_t clientID);
00107
00108         private:
00110             std::shared_ptr<Process::Manager> _manager;
00112             std::shared_ptr<Process::WorkerController> _listener;
00113         };
00114     }
00115 }
00116
00117 #endif /* __BE_PROCESS_MESSAGECENTER__ */
```

I.102 be_process_posixthreadmanager.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_POSIXTHREADMANAGER_H__
00012 #define __BE_PROCESS_POSIXTHREADMANAGER_H__
00013
00014 #include <pthread.h>
00015
00016 #include <be_process_manager.h>
00017 #include <be_process_workercontroller.h>
00018
00019 namespace BiometricEvaluation
00020 {
00021     namespace Process
00022     {
00023         /* Forward declaration */
00024         class POSIXThreadWorkerController;
00025
00031         class POSIXThreadManager : public Manager
00032         {
00033         public:
00037             POSIXThreadManager();
00038
00049             std::shared_ptr<WorkerController>
00050             addWorker(
00051                 std::shared_ptr<Worker> worker);
00052
00069             void
00070             startWorkers(
00071                 bool wait = true,
00072                 bool communicate = false);
00073
00093             void
00094             startWorker(
00095                 std::shared_ptr<WorkerController> worker,
00096                 bool wait = true,
00097                 bool communicate = false);
00098
00112             void
00113             stopWorker(
00114                 std::shared_ptr<WorkerController> workerController);
00115
00116             void
00117             waitForWorkerExit();
00118
00123             ~POSIXThreadManager();
00124
00125         private:
00130             void
00131             _wait();
00132         };
00133
00138         class POSIXThreadWorkerController : public WorkerController
```

```
00139         {
00140         public:
00149             void
00150             reset();
00151
00159             bool
00160             isWorking()
00161                 const;
00162
00163             bool
00164             everWorked()
00165                 const;
00166
00171             ~POSIXThreadWorkerController();
00172
00173         protected:
00174
00175         private:
00183             POSIXThreadWorkerController(
00184                 std::shared_ptr<Worker> worker);
00185
00203             void
00204             start(
00205                 bool communicate = false);
00206
00216             void
00217             stop();
00218
00230             static void *
00231             workerMainWrapper(
00232                 void *_this);
00233
00234             /*
00235              * Friends.
00236              * XXX We just need addWorker(), startWorkers(), and
00237              * XXX _wait() to be friended, but you cannot friend
00238              * XXX a private function (_wait()).
00239              * TODO Eliminate _wait() from WorkerController?
00240              */
00241             friend class POSIXThreadManager;
00242
00244             pthread_t _thread;
00245
00247             bool _working;
00248
00250             bool _hasWorked;
00251         };
00252     }
00253 }
00254
00255 #endif /* __BE_PROCESS_POSIXTHREADMANAGER_H__ */
```

I.103 be_process_semaphore.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_SEMAPHORE_H__
00012 #define __BE_PROCESS_SEMAPHORE_H__
00013
00014 #include <sys/types.h>
00015 #include <semaphore.h>
00016 #include <stdint.h>
00017
00018 #include <string>
00019
00020 namespace BiometricEvaluation
00021 {
```

```
00022     namespace Process
00023     {
00042         class Semaphore
00043         {
00044         public:
00067             Semaphore(
00068                 const std::string &name,
00069                 const mode_t mode,
00070                 const int value,
00071                 const bool force = false);
00072
00084             Semaphore(
00085                 const std::string &name);
00086
00087             ~Semaphore();
00088
00102             bool wait(const bool interruptible);
00103
00117             bool trywait(const bool interruptible);
00118
00138             bool timedwait(
00139                 const uint64_t interval,
00140                 const bool interruptible);
00141
00150             void post();
00151
00159             std::string getName();
00160
00161         protected:
00162
00163         private:
00165             sem_t * _semaphore;
00166             std::string _name;
00167             pid_t _creatorPID;
00168         };
00169     }
00170 }
00171
00172 #endif /* __BE_PROCESS_SEMAPHORE_H__ */
00173
```

I.104 be_process_statistics.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_STATISTICS_H__
00012 #define __BE_PROCESS_STATISTICS_H__
00013
00014 #include <memory>
00015 #include <optional>
00016 #include <tuple>
00017
00018 #include <be_io_autologger.h>
00019
00020 namespace BiometricEvaluation {
00021     namespace Process {
00022
00043         class Statistics {
00044         public:
00045
00050             Statistics();
00051
00074             Statistics(const std::shared_ptr<IO::FileLogCabinet>
00075                 &logCabinet, bool doTasksLogging = false);
00076
00093             Statistics(
00094                 const std::shared_ptr<IO::Logsheet> &logSheet,
```

```
00095                 std::optional<std::shared_ptr<IO::Logsheet>>
00096                 tasksLogSheet = std::nullopt);
00097
00098             ~Statistics();
00099
00124             std::tuple<
00125                 uint64_t,
00126                 uint64_t> getCPUTimes();
00127
00156             std::vector<std::tuple<
00157                 pid_t,
00158                 float,
00159                 float>> getTasksStats();
00160
00185             std::tuple<
00186                 uint64_t,
00187                 uint64_t,
00188                 uint64_t,
00189                 uint64_t,
00190                 uint64_t> getMemorySizes();
00191
00205             uint32_t getNumThreads();
00206
00222             void logStats();
00223
00231             std::string
00232             getComment()
00233                 const;
00234
00244             void
00245             setComment(
00246                 std::string_view comment);
00247
00255             void startAutoLogging(
00256                 std::chrono::microseconds interval);
00257             [[deprecated("Use std::chrono values instead")]]
00258             void startAutoLogging(uint64_t interval);
00259
00266             void stopAutoLogging();
00267
00268         private:
00269             IO::AutoLogger _autoLogger{};
00270             IO::AutoLogger _autoTaskLogger{};
00271             pid_t _pid{};
00272             std::shared_ptr<IO::FileLogCabinet> _logCabinet{};
00273             std::shared_ptr<IO::Logsheet> _logSheet{};
00274             std::optional<std::shared_ptr<IO::Logsheet>>
00275                 _tasksLogSheet{};
00276             pid_t _loggingTaskID{};
00277             pid_t _taskLoggingTaskID{};
00278             std::string getStatsLogEntry() const;
00279             std::string getTasksStatsLogEntry() const;
00280             bool _doTasksLogging{};
00281             bool _logging{};
00282         };
00283
00284     }
00285 }
00286 #endif /* __BE_PROCESS_STATISTICS_H__ */
```

I.105 be_process_worker.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_WORKER_H__
00012 #define __BE_PROCESS_WORKER_H__
00013
```

```
00014 #include <cstdint>
00015
00016 #include <be_error_exception.h>
00017 #include <be_memory_autoarray.h>
00018 #include <be_process.h>
00019
00020 namespace BiometricEvaluation
00021 {
00022     namespace Process
00023     {
00029         class Worker
00030         {
00031         public:
00051             virtual int32_t
00052             workerMain() = 0;
00053
00067             std::shared_ptr<void>
00068             getParameter(
00069                 const std::string &name);
00070
00085             double
00086             getParameterAsDouble(
00087                 const std::string &name);
00088
00103             int64_t
00104             getParameterAsInteger(
00105                 const std::string &name);
00106
00121             std::string
00122             getParameterAsString(
00123                 const std::string &name);
00124
00134             void
00135             setParameter(
00136                 const std::string &name,
00137                 std::shared_ptr<void> argument);
00138
00143             virtual void
00144             stop()
00145                 final;
00146
00158             void
00159             closeWorkerPipeEnds();
00160
00172             void
00173             closeManagerPipeEnds();
00174
00188             int
00189             getSendingPipe() const;
00190
00204             int
00205             getReceivingPipe() const;
00206
00219             void
00220             sendMessageToManager(
00221                 const Memory::uint8Array &message);
00222
00236             void
00237             receiveMessageFromManager(
00238                 Memory::uint8Array &message);
00239
00248             void
00249             _initCommunication();
00250
00255             virtual ~Worker();
00256
00257         protected:
00262             Worker();
00263
00272             virtual bool
00273             stopRequested()
00274                 const
00275                 final;
00276
00289             bool
00290             waitForMessage(
00291                 int numSeconds = -1)
```

```
00292                 const;
00293
00294         private:
00296             volatile bool _stopRequested;
00297
00299             ParameterList _parameters;
00300
00302             bool _communicationEnabled;
00304             int _pipeToChild[2];
00306             int _pipeFromChild[2];
00307         };
00308     }
00309 }
00310
00311 #endif /* __BE_PROCESS_WORKER_H__ */
```

I.106 be_process_workercontroller.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_PROCESS_WORKERCONTROLLER_H__
00012 #define __BE_PROCESS_WORKERCONTROLLER_H__
00013
00014 #include <memory>
00015
00016 #include <be_error_exception.h>
00017 #include <be_memory_autoarray.h>
00018 #include <be_process.h>
00019 #include <be_process_worker.h>
00020
00021 namespace BiometricEvaluation
00022 {
00023     namespace Process
00024     {
00029         class WorkerController
00030         {
00031         public:
00038             WorkerController(
00039                 std::shared_ptr<Worker> worker);
00040
00055             virtual void
00056             sendMessageToWorker(
00057                 const Memory::uint8Array &message);
00058
00073             virtual void
00074             setParameter(
00075                 const std::string &name,
00076                 std::shared_ptr<void> argument);
00077
00092             virtual void
00093             setParameterFromDouble(
00094                 const std::string &name,
00095                 double argument);
00096
00111             virtual void
00112             setParameterFromInteger(
00113                 const std::string &name,
00114                 int64_t argument);
00115
00130             virtual void
00131             setParameterFromString(
00132                 const std::string &name,
00133                 const std::string &argument);
00134
00143             virtual void
00144             reset();
00145
```

```
00153             virtual bool
00154             isWorking()
00155                 const = 0;
00156
00168             virtual bool
00169             everWorked()
00170                 const = 0;
00171
00184             inline bool
00185             finishedWorking()
00186                 const
00187             {
00188                 return (this->everWorked() &&
00189                     !this->isWorking());
00190             }
00191
00199             std::shared_ptr<Worker>
00200             getWorker()
00201                 const;
00202
00216             virtual int32_t
00217             getExitStatus()
00218                 const
00219                 final;
00220
00225             virtual ~WorkerController();
00226
00227         protected:
00229             std::shared_ptr<Worker> _worker;
00231             bool _rvSet;
00233             int32_t _rv;
00234
00235         private:
00253             virtual void
00254             start(
00255                 bool communicate = false) = 0;
00256
00266             virtual void
00267             stop() = 0;
00268         };
00269     }
00270 }
00271
00272 #endif /* __BE_PROCESS_WORKERCONTROLLER_H__ */
```

I.107 be_sysdeps.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef BE_WINDOWSFIXUP_H_
00012 #define BE_WINDOWSFIXUP_H_
00013
00014 #ifdef _WIN32
00015
00016 #include <windows.h>
00017
00018 using mode_t = unsigned int;
00019
00020 /* 1: arpa/inet.h equivalent for htons, etc. */
00021 #include <winsock.h>
00022
00023 /* 2: use windows basename(3)/dirname(3) */
00024 char* basename(const char*);
00025 char* dirname(const char*);
00026
00027 /* 3: unistd.h */
00028 #include <io.h>
```

```
00029
00030 /* Windows only implements localtime and localtime_s, but _s is better */
00031 struct tm*
00032 localtime_r(
00033     const time_t* clock,
00034     struct tm* result);
00035
00036 /* 4: Mainly in windows.h */
00037
00038 /*
00039  * 5: No dirent on Windows. Use open-source version compiled in NBIS.
00040  *    Also contains some #defines usually found in sys/stat.h.
00041  */
00042 #include <be_dirent_windows.h>
00043 #ifndef S_IRWXU
00044 #define S_IRWXU S_IRUSR | S_IWUSR | S_IXUSR
00045 #endif
00046 #ifndef S_IRWXG
00047 #define S_IRWXG S_IRGRP | S_IWGRP | S_IXGRP
00048 #endif
00049 #ifndef S_IRWXO
00050 #define S_IRWXO S_IROTH | S_IWOTH | S_IXOTH
00051 #endif
00052
00053 #ifndef F_OK
00054 #define F_OK 00
00055 #endif
00056
00057 #ifndef R_OK
00058 #define R_OK 04
00059 #endif
00060
00061 #ifndef W_OK
00062 #define W_OK 02
00063 #endif
00064
00065 /* 6: mkdir() */
00066 #include <direct.h>
00067 /* ...except Windows mkdir doesn't take a mode_t */
00068 int mkdir(const char*, mode_t);
00069 int mkstemp(char*);
00070
00071 /* 7: Symlinks aren't quite the same on Windows */
00072 int lstat(const char*, struct stat*);
00073
00074 /* 8: Missing or alternate names with required C linkage for NBIS */
00075
00076 #ifdef __cplusplus
00077 extern "C" {
00078 #endif
00079
00080 int strncasecmp(const char* s1, const char* s2, size_t n);
00081 char* index(const char* s, int c);
00082
00083 /* 9: No gettimeofday */
00084 int gettimeofday(struct timeval*, struct timezone*);
00085
00086 /* 10: Skipping for now (sys/wait.h) */
00087
00088 #ifdef __cplusplus
00089 }
00090 #endif
00091
00092 #else
00093
00094 /* 1 */
00095 #include <arpa/inet.h>
00096
00097 /* 2 */
00098
00099 #include <libgen.h> /*  for basename(3) and dirname(3) */
00100 #ifdef basename     /* GNU has this macro irresponsibly defined */
00101 #undef basename
00102 #endif
00103
00104 /* 3 */
00105 #include <unistd.h>
```

```
00106
00107 /* 4 */
00108 #include <sys/param.h>
00109
00110 /* 5 */
00111 #include <dirent.h>
00112
00113 /* 9 */
00114 #include <sys/time.h>
00115
00116 /* 10 */
00117 #include <sys/wait.h>
00118
00119 #endif /* _WIN32 */
00120
00121 #endif /* BE_WINDOWSFIXUP_H_ */
```

I.108 be_system.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_SYSTEM_H__
00012 #define __BE_SYSTEM_H__
00013
00014 #include <cstdint>
00015 #include <map>
00016 #include <string>
00017
00018 namespace BiometricEvaluation
00019 {
00027     namespace System
00028     {
00039         uint32_t getCPUCount();
00040
00049         uint32_t getCPUCoreCount();
00050
00063         uint32_t getCPUSocketCount();
00064
00074         uint32_t getCPUCoreCount();
00075
00084         uint64_t getRealMemorySize();
00085
00087         std::map<std::string, uint64_t> getMemInfo();
00088
00097         double getLoadAverage();
00098     }
00099 }
00100 #endif /* __BE_SYSTEM_H__ */
```

I.109 be_system_memlog.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_SYSTEM_MEMLOG_H__
00012 #define __BE_SYSTEM_MEMLOG_H__
00013
```

```
00014 #include <chrono>
00015 #include <cstdint>
00016 #include <map>
00017 #include <string_view>
00018
00019 #include <be_io_autologger.h>
00020 #include <be_system.h>
00021
00022 namespace BiometricEvaluation
00023 {
00031     namespace System
00032     {
00033         class MemoryLogger {
00034         public:
00041             MemoryLogger(
00042                 const std::shared_ptr<IO::Logsheet> &logSheet);
00043
00044             ~MemoryLogger();
00045
00053             std::string getComment() const;
00054
00062             void addLogEntry();
00063
00073             void setComment(std::string_view comment);
00074
00089             void startAutoLogging(
00090                                 std::chrono::microseconds interval,
00091                 bool writeHeader = true);
00099             void stopAutoLogging();
00100
00101         private:
00102             IO::AutoLogger _autoLogger{};
00103             std::shared_ptr<IO::Logsheet> _logSheet{};
00108             std::string getMemLogEntry();
00109         };
00110     }
00111 }
00112 #endif /* __BE_SYSTEM_MEMLOG_H__ */
```

I.110 be_text.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010 #ifndef __BE_TEXT_H__
00011 #define __BE_TEXT_H__
00012
00013 #include <locale>
00014 #include <string>
00015 #include <vector>
00016
00017 #include <be_error_exception.h>
00018 #include <be_memory_autoarray.h>
00019
00020 namespace BiometricEvaluation {
00021
00030     namespace Text {
00031
00046         std::string
00047         trimWhitespace(
00048             const std::string &s,
00049             const std::locale &locale = std::locale());
00050
00064         std::string
00065         ltrimWhitespace(
00066             const std::string &s,
00067             const std::locale &locale = std::locale());
00068
00082         std::string
```

```
00083         rtrimWhitespace(
00084             const std::string &s,
00085             const std::locale &locale = std::locale());
00086
00100         std::string
00101         trim(
00102             const std::string &s,
00103             const char trimChar);
00104
00117         std::string
00118         ltrim(
00119             const std::string &s,
00120             const char trimChar);
00121
00134         std::string
00135         rtrim(
00136             const std::string &s,
00137             const char trimChar);
00138
00160         std::string
00161         digest(
00162             const std::string &s,
00163             const std::string &digest = "md5");
00164
00188         std::string
00189         digest(
00190             const void *buffer,
00191             const size_t buffer_size,
00192             const std::string &digest = "md5");
00193
00215         std::vector<std::string>
00216         split(
00217             const std::string &str,
00218             const char delimiter,
00219             bool escape = true);
00220
00234         std::string
00235         basename(
00236             const std::string &path);
00237
00250         std::string
00251         dirname(
00252             const std::string &path);
00253
00267         bool
00268         caseInsensitiveCompare(
00269             const std::string &str1,
00270             const std::string &str2);
00271
00284         std::string
00285         toUppercase(
00286             const std::string &str,
00287             const std::locale &locale = std::locale());
00288
00301         std::string
00302         toLowercase(
00303             const std::string &str,
00304             const std::locale &locale = std::locale());
00305
00316         std::string
00317         encodeBase64(
00318             const BiometricEvaluation::Memory::uint8Array &data);
00319
00330         BiometricEvaluation::Memory::uint8Array
00331         decodeBase64(
00332             const std::string &data);
00333     }
00334 }
00335 #endif /* __BE_TEXT_H__ */
```

I.111 be_time.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
```

```
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_TIME_H__
00012 #define __BE_TIME_H__
00013
00014 #include <cstdint>
00015
00016 #include <be_error_exception.h>
00017
00018 namespace BiometricEvaluation
00019 {
00027     namespace Time
00028     {
00029         const uint64_t OneSecond = 1000000;
00030         const uint64_t OneHalfSecond = 500000;
00031         const uint64_t OneQuarterSecond = 250000;
00032         const uint64_t OneEighthSecond = 125000;
00033         const int NanosecondsPerMicrosecond = 1000;
00034         const int MicrosecondsPerSecond = 1000000;
00035         const int MicrosecondsPerMillisecond = 1000;
00036         const int MillisecondsPerSecond = 1000;
00037
00039         std::string
00040         getCurrentTime();
00041
00043         std::string
00044         getCurrentDate();
00045
00050         std::string
00051         getCurrentDateAndTime();
00052
00066         std::string
00067         getCurrentCalendarInformation(
00068             const std::string &formatString);
00069
00078         std::string
00079         put_time(
00080             const struct tm *tmb,
00081             const char *fmt);
00082
00083     }
00084 }
00085 #endif /* __BE_TIME_H__ */
```

I.112 be_time_timer.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_TIME_TIMER_H__
00012 #define __BE_TIME_TIMER_H__
00013
00014 #include <chrono>
00015 #include <cstdint>
00016 #include <functional>
00017
00018 #include <be_time.h>
00019
00020 namespace BiometricEvaluation
00021 {
00022     namespace Time
00023     {
00038         class Timer
```

```
00039         {
00040         public:
00042             using BE_CLOCK_TYPE =
00043 #ifdef __MIC__
00044                 std::chrono::monotonic_clock;
00045 #else
00046                 std::chrono::steady_clock;
00047 #endif
00048
00049             /* Ensure chosen clock increases monotonically */
00050             static_assert(BE_CLOCK_TYPE::is_steady,
00051                 "BE_CLOCK_TYPE is not a steady clock");
00052
00054             Timer();
00055
00066             Timer(
00067                 const std::function<void()> &func);
00068
00077             void
00078             start();
00079
00088             void
00089             stop();
00090
00122             template<typename Duration>
00123             std::uintmax_t
00124             elapsed()
00125                 const
00126             {
00127                 return (std::chrono::duration_cast<Duration>(
00128                     this->elapsedTimePoint()).count());
00129             }
00130
00149             template<typename Duration>
00150             std::string
00151             elapsedStr(
00152                 bool displayUnits = false)
00153                 const
00154             {
00155                 const std::string ret{std::to_string(
00156                     this->elapsed<Duration>())};
00157
00158                 if (displayUnits)
00159                     return (ret + Timer::units<Duration>());
00160                 return (ret);
00161             }
00162
00171             template<typename Duration>
00172             static
00173             std::string
00174             units()
00175             {
00176                 if ((Duration::period::num ==
00177                     std::chrono::nanoseconds::period::num) &&
00178                     (Duration::period::den ==
00179                     std::chrono::nanoseconds::period::den)) {
00180                     return ("ns");
00181                 } else if ((Duration::period::num ==
00182                     std::chrono::microseconds::period::num) &&
00183                     (Duration::period::den ==
00184                     std::chrono::microseconds::period::den)) {
00185                     return ("\xC2\xB5s");
00186                 } else if ((Duration::period::num ==
00187                     std::chrono::milliseconds::period::num) &&
00188                     (Duration::period::den ==
00189                     std::chrono::milliseconds::period::den)) {
00190                     return ("ms");
00191                 }  else if ((Duration::period::num ==
00192                     std::chrono::seconds::period::num) &&
00193                     (Duration::period::den ==
00194                     std::chrono::seconds::period::den)) {
00195                     return ("s");
00196                 } else if ((Duration::period::num ==
00197                     std::chrono::minutes::period::num) &&
00198                     (Duration::period::den ==
00199                     std::chrono::minutes::period::den)) {
00200                     return ("m");
```

```
00201                 } else if ((Duration::period::num ==
00202                     std::chrono::hours::period::num) &&
00203                     (Duration::period::den ==
00204                     std::chrono::hours::period::den)) {
00205                     return ("h");
00206                 } else {
00207                     throw BiometricEvaluation::Error::
00208                         StrategyError{"Unknown duration "
00209                         "units"};
00210                 }
00211             }
00212
00236             std::common_type_t<BE_CLOCK_TYPE::time_point::duration,
00237                 BE_CLOCK_TYPE::time_point::duration>
00238             elapsedTimePoint()
00239                 const;
00240
00255             Timer&
00256             time(
00257                 const std::function<void()> &func);
00258
00259         private:
00264             bool _inProgress;
00265
00267             BE_CLOCK_TYPE::time_point _start;
00269             BE_CLOCK_TYPE::time_point _finish;
00270         };
00271
00288         std::ostream&
00289         operator<<(
00290             std::ostream &s,
00291             const Timer &timer);
00292     }
00293 }
00294
00295 #endif /* __BE_TIME_TIMER_H__ */
```

I.113 be_time_watchdog.h

```
00001 /******************************************************************************
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  *****************************************************************************/
00010 #ifndef __BE_TIME_WATCHDOG_H__
00011 #define __BE_TIME_WATCHDOG_H__
00012
00013 #include <csetjmp>
00014 #include <csignal>
00015
00016 #include <be_time.h>
00017 #include <be_error_exception.h>
00018
00025 #define BEGIN_WATCHDOG_BLOCK(_watchdog, _blockname) do {        \
00026     if (!(_watchdog)->isEnabled())                              \
00027         break;                          \
00028     (_watchdog)->clearExpired();                                \
00029     (_watchdog)->clearCanSigJump();                             \
00030     if (sigsetjmp(                          \
00031         BiometricEvaluation::Time::Watchdog::_sigJumpBuf, 1) != 0) \
00032      {                                  \
00033         (_watchdog)->setExpired();              \
00034         goto _blockname ## _end;                \
00035     }                                   \
00036     (_watchdog)->setCanSigJump();                               \
00037     (_watchdog)->start();                           \
00038 } while (0)
00039
00040 #define END_WATCHDOG_BLOCK(_watchdog, _blockname) do {          \
00041     if (!(_watchdog)->isEnabled())                              \
00042         break;                          \
```

```
00043     _blockname ## _end:                     \
00044     (_watchdog)->clearCanSigJump();               \
00045     (_watchdog)->stop();                        \
00046 } while (0);
00047
00048 #define ABORT_WATCHDOG(_watchdog) do {                  \
00049     if (!(_watchdog)->isEnabled())                \
00050         break;                              \
00051     (_watchdog)->clearCanSigJump();               \
00052     (_watchdog)->stop();                        \
00053 } while (0);
00054
00055 namespace BiometricEvaluation {
00056
00057     namespace Time {
00058
00118         class Watchdog {
00119         public:
00120
00122             static const uint8_t PROCESSTIME = 0;
00124             static const uint8_t REALTIME = 1;
00125
00141             Watchdog(const uint8_t type);
00142
00150             uint64_t
00151             getInterval()
00152                 const
00153                 noexcept;
00154
00166             void setInterval(uint64_t interval);
00167
00175             void start();
00176
00183             void stop();
00184
00189             bool expired();
00190
00195             void setCanSigJump();
00196
00201             void clearCanSigJump();
00202
00206             void setExpired();
00207
00211             void clearExpired();
00212
00223             void setEnabled(
00224                 const bool enabled);
00225
00227             bool isEnabled() const;
00228
00229             /*
00230              * Flag indicating can jump after handling a signal,
00231              * and the jump buffer used by the signal handler.
00232              */
00233             static bool _canSigJump;
00234             static sigjmp_buf _sigJumpBuf;
00235
00236         protected:
00237
00238         private:
00239             /*
00240              * Definition of the signal handler, a class method
00241              * that will handle all signals managed by this object,
00242              * conditionally jumping to a jump block. This jump
00243              * capability allows applications to bypass code that
00244              * is "hung". Applications should use the
00245              * BEGIN_WATCHDOG_BLOCK()/END_WATCHDOG_BLOCK() macro
00246              * pair to take advantage of this capability.
00247              */
00248             static void sighandler(int signo);
00249
00251             bool _enabled{true};
00252
00253             /*
00254              * Current timer interval.
00255              */
00256             uint64_t _interval;
```

```
00257
00258             /*
00259              * The type of timer.
00260              */
00261             uint8_t _type;
00262
00263             /*
00264              * Flag indicated that the timer expired.
00265              */
00266             bool _expired;
00267
00268             /*
00269              * Utility function to map the Watchdog type of alarm
00270              * to the system signal number and which system timer.
00271              */
00272             void internalMapWatchdogType(int *signo, int *which);
00273         };
00274         /*
00275          * Declaration of the signal handler, a function with C linkage
00276          * that will handle the alarm signals sent when a system timer
00277          * expires.
00278          */
00279         extern "C" {
00280             void WatchdogSignalHandler(int signo, siginfo_t *info,
00281                 void *uap);
00282         }
00283     }
00284 }
00285 #endif  /* __BE_TIME_WATCHDOG_H__ */
```

I.114 be_video.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_VIDEO_H__
00012 #define __BE_VIDEO_H__
00013
00014 #include <be_framework_enumeration.h>
00015 #include <be_image.h>
00016 #include <be_memory_autoarray.h>
00017
00018 namespace BiometricEvaluation
00019 {
00028     namespace Video
00029     {
00031         enum class CodingFormat
00032         {
00033             None        = 0,
00034             MPEG1       = 1,
00035             MPEG2       = 2,
00036             MPEG4       = 3,
00037             H264        = 4
00038         };
00039
00041         enum class ContainerFormat
00042         {
00043             MPEG1PS     = 1,
00044             MPEG2TS     = 2,
00045             MPEG4PS     = 3,
00046             AVI     = 4
00047         };
00048         struct Frame {
00049             Image::Size size;
00050             int64_t timestamp;
00051             Memory::uint8Array data;
00052         };
00053     }
```

```
00054 }
00055 #endif /* __BE_VIDEO_H__ */
```

I.115 be_video_container.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_VIDEO_CONTAINER_H__
00012 #define __BE_VIDEO_CONTAINER_H__
00013
00014 #include <memory.h>
00015 #include <be_video_stream.h>
00016
00017 namespace BiometricEvaluation
00018 {
00019     namespace Video
00020     {
00029         class Container {
00030         public:
00042             Container(const Memory::uint8Array &buffer);
00043
00056             Container(
00057                 const std::shared_ptr<Memory::uint8Array> &buffer);
00058
00069             Container(const std::string &filename);
00070
00075             uint32_t getAudioCount();
00076
00081             uint32_t getVideoCount();
00082
00094             std::unique_ptr<Video::Stream>
00095                 getVideoStream(uint32_t videoNum);
00096
00097             ~Container();
00098
00099             class Impl;
00100         private:
00101             std::unique_ptr<Container::Impl> pimpl;
00102         };
00103     }
00104 }
00105 #endif /* __BE_VIDEO_CONTAINER_H__ */
```

I.116 be_video_stream.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_VIDEO_STREAM_H
00012 #define __BE_VIDEO_STREAM_H
00013
00014 #include <be_image.h>
00015 #include <be_video.h>
00016 namespace BiometricEvaluation
00017 {
00023     namespace Video
00024     {
```

```
00025         class Stream {
00026         public:
00034             virtual float getFPS() = 0;
00035
00043             virtual uint64_t getFrameCount() = 0;
00044
00060             virtual Video::Frame getFrame(
00061                 uint32_t frameNum) = 0;
00062
00080 #undef PixelFormat
00081             virtual std::vector<Video::Frame> getFrameSequence(
00082                 int64_t startTime,
00083                 int64_t endTime) = 0;
00084
00095             virtual void setFrameScale(
00096                 float xScale,
00097                 float yScale) = 0;
00098
00107             virtual void setFramePixelFormat(
00108                 const Image::PixelFormat pixelFormat) = 0;
00109
00110             virtual ~Stream();
00111         };
00112     }
00113 }
00114 #endif /* __BE_VIDEO_STREAM_H */
```

I.117 be_view.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_VIEW_H__
00012 #define __BE_VIEW_H__
00013
00014 namespace BiometricEvaluation
00015 {
00024     namespace View
00025     {
00026     }
00027 }
00028 #endif /* __BE_VIEW_H__ */
```

I.118 be_view_an2kview.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_VIEW_AN2KVIEW_H__
00012 #define __BE_VIEW_AN2KVIEW_H__
00013
00014 #include <set>
00015 #include <string>
00016 #include <vector>
00017
00018 #include <memory>
00019
00020 #include <be_finger_an2kminutiae_data_record.h>
```

```
00021 #include <be_framework_enumeration.h>
00022 #include <be_memory_autobuffer.h>
00023 #include <be_view_view.h>
00024 #include <be_image_image.h>
00025
00026 /* an2k.h forward declares */
00027 struct record;
00028 typedef record RECORD;
00029 struct ansi_nist;
00030 typedef ansi_nist ANSI_NIST;
00031
00032 namespace BiometricEvaluation
00033 {
00034     namespace View
00035     {
00051         class AN2KView : public BiometricEvaluation::View::View {
00052         public:
00054             enum class RecordType : uint16_t
00055             {
00056                 Type_1 = 1,
00057                 Type_2 = 2,
00058                 Type_3 = 3,
00059                 Type_4 = 4,
00060                 Type_5 = 5,
00061                 Type_6 = 6,
00062                 Type_7 = 7,
00063                 Type_8 = 8,
00064                 Type_9 = 9,
00065                 Type_10 = 10,
00066                 Type_11 = 11,
00067                 Type_12 = 12,
00068                 Type_13 = 13,
00069                 Type_14 = 14,
00070                 Type_15 = 15,
00071                 Type_16 = 16,
00072                 Type_17 = 17,
00073                 Type_99 = 99
00074             };
00075
00081             enum class DeviceMonitoringMode
00082             {
00087                 Controlled,
00093                 Assisted,
00099                 Observed,
00104                 Unattended,
00106                 Unknown,
00108                 NA
00109             };
00110
00126             static DeviceMonitoringMode
00127             convertDeviceMonitoringMode(
00128                 const char *dmm);
00129
00149             static Image::CompressionAlgorithm
00150                 convertCompressionAlgorithm(
00151                 const uint16_t recordType,
00152                 const unsigned char *an2kValue);
00153
00159             static const double MinimumScanResolutionPPMM;
00160             static const double HalfMinimumScanResolutionPPMM;
00161
00166             static const int FixedResolutionBitDepth = 8;
00167
00175             AN2KView(
00176                 const std::string filename,
00177                 const RecordType typeID,
00178                 const uint32_t recordNumber);
00179
00187             AN2KView(
00188                 Memory::uint8Array &buf,
00189                 const RecordType typeID,
00190                 const uint32_t recordNumber);
00191
00192             ~AN2KView();
00193
00204             std::vector<Finger::AN2KMinutiaeDataRecord>
00205             getMinutiaeDataRecordSet() const;
```

```
00206
00213             RecordType getRecordType() const;
00214
00216             int
00217             getIDC()
00218                 const;
00219
00220         protected:
00221
00226             Memory::AutoBuffer<ANSI_NIST>
00227             getAN2K()
00228                 const;
00229
00238             RECORD*
00239             getAN2KRecord()
00240                 const;
00241
00242         private:
00243
00262             void readImageCommon(
00263                 const ANSI_NIST *an2k,
00264                 const RecordType typeID,
00265                 const uint32_t recordNumber);
00266
00275             void
00276             associateMinutiaeData(
00277                 Memory::uint8Array &buf);
00278
00287                 void
00288             associateMinutiaeData(
00289                 const std::string &filename);
00290
00299             void
00300             addMinutiaeDataRecord(
00301                 Finger::AN2KMinutiaeDataRecord &mdr);
00302
00303             /* The record that this object represents. The Nth
00304              * record is searched for when the object is
00305              * constructed and may be referenced by subclasses.
00306              */
00307             Memory::AutoBuffer<ANSI_NIST> _an2k;
00308             RECORD *_an2kRecord;
00309             RecordType _recordType;
00310             int _idc;
00311
00316             std::vector<Finger::AN2KMinutiaeDataRecord>
00317                 _minutiaeDataRecordSet;
00318         };
00319
00324         std::ostream&
00325         operator<<(
00326             std::ostream &stream,
00327             const AN2KView::DeviceMonitoringMode &kind);
00328     }
00329 }
00330
00331 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00332     BiometricEvaluation::View::AN2KView::RecordType,
00333     BE_View_AN2KView_RecordType_EnumToStringMap);
00334
00335 BE_FRAMEWORK_ENUMERATION_DECLARATIONS(
00336     BiometricEvaluation::View::AN2KView::DeviceMonitoringMode,
00337     BE_View_AN2KView_DeviceMonitoringMode_EnumToStringMap);
00338
00339 #endif /* __BE_VIEW_AN2KVIEW_H__ */
00340
```

I.119 be_view_an2kview_varres.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
```

```
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_VIEW_AN2KVIEW_VARRES_H__
00012 #define __BE_VIEW_AN2KVIEW_VARRES_H__
00013
00014 #include <iostream>
00015 #include <map>
00016 #include <string>
00017
00018 #include <be_error_exception.h>
00019 #include <be_feature.h>
00020 #include <be_view_an2kview.h>
00021
00022 /* an2k.h forward declares */
00023 struct field;
00024 typedef field FIELD;
00025
00026 namespace BiometricEvaluation
00027 {
00028     namespace View
00029     {
00038         class AN2KViewVariableResolution : public AN2KView {
00039         public:
00049             struct AN2KQualityMetric
00050             {
00051                 Feature::FGP        fgp;
00052                 uint8_t         score;
00053                 uint16_t        vendorID;
00054                 uint16_t        productCode;
00055             };
00056             using AN2KQualityMetric = struct AN2KQualityMetric;
00057             using QualityMetricSet = std::vector<AN2KQualityMetric>;
00058
00064             struct PrintPositionCoordinate {
00066                 Finger::FingerImageCode fingerView;
00068                 Finger::FingerImageCode segment;
00070                 Image::CoordinateSet coordinates;
00071             };
00072             using PrintPositionCoordinate =
00073                 struct PrintPositionCoordinate;
00074             using PrintPositionCoordinateSet =
00075                 std::vector<PrintPositionCoordinate>;
00076
00090             static QualityMetricSet
00091             extractQuality(FIELD *field, Feature::PositionType type);
00092
00097             Finger::Impression getImpressionType() const;
00098
00103             std::string getSourceAgency() const;
00104
00109             std::string getCaptureDate() const;
00110
00119             std::string getComment() const;
00120
00128             Finger::CaptureTechnology
00129             getCaptureTechnology()
00130                 const;
00131
00152             Memory::uint8Array
00153             getUserDefinedField(
00154                 const uint16_t field) const;
00155
00175             static Memory::uint8Array
00176             parseUserDefinedField(
00177                 const RECORD* const record,
00178                 int fieldID);
00179
00195             static Finger::CaptureTechnology
00196             convertCaptureTechnology(
00197                 const char *str);
00198
00199         protected:
00207             AN2KViewVariableResolution(
00208                 const std::string &filename,
00209                 const RecordType typeID,
```

```
00210                 const uint32_t recordNumber);
00211
00220             AN2KViewVariableResolution(
00221                 Memory::uint8Array &buf,
00222                 const RecordType typeID,
00223                 const uint32_t recordNumber);
00224
00238                         Feature::FGPSet
00239                         getPositions() const;
00240
00251             Finger::PositionDescriptors
00252             getPositionDescriptors()
00253                 const;
00254
00262             PrintPositionCoordinateSet
00263             getPrintPositionCoordinates()
00264                 const;
00265
00273             QualityMetricSet
00274             getQualityMetric()
00275                 const;
00276
00277         private:
00278             void readImageRecord(
00279                 const RecordType typeID);
00280
00281             Feature::FGPSet _positions;
00282             Finger::Impression _imp;
00283             Finger::CaptureTechnology _frct{
00284                 Finger::CaptureTechnology::Unknown};
00285             std::string _sourceAgency;
00286             std::string _captureDate;
00287             std::string _comment;
00288             Finger::PositionDescriptors _pd;
00289             PrintPositionCoordinateSet _ppcs;
00291             QualityMetricSet _qms;
00293             mutable std::map<uint16_t, Memory::uint8Array> _udf;
00294         };
00295
00309         std::ostream&
00310         operator<<(
00311             std::ostream &s,
00312             const AN2KViewVariableResolution::AN2KQualityMetric &qm);
00313
00327         std::ostream&
00328         operator<<(
00329             std::ostream &stream,
00330             const AN2KViewVariableResolution::PrintPositionCoordinate
00331             &ppc);
00332     }
00333 }
00334 #endif /* __BE_VIEW_AN2KVIEW_VARRES_H__ */
00335
```

I.120 be_view_view.h

```
00001 /*
00002  * This software was developed at the National Institute of Standards and
00003  * Technology (NIST) by employees of the Federal Government in the course
00004  * of their official duties. Pursuant to title 17 Section 105 of the
00005  * United States Code, this software is not subject to copyright protection
00006  * and is in the public domain. NIST assumes no responsibility whatsoever for
00007  * its use by other parties, and makes no guarantees, expressed or implied,
00008  * about its quality, reliability, or any other characteristic.
00009  */
00010
00011 #ifndef __BE_VIEW_VIEW_H__
00012 #define __BE_VIEW_VIEW_H__
00013
00014 #include <memory>
00015 #include <string>
00016 #include <vector>
00017
00018 #include <be_image_image.h>
00019
```

```
00020 namespace BiometricEvaluation
00021 {
00022     namespace View
00023     {
00031         class View {
00032         public:
00033
00045             std::shared_ptr<Image::Image>
00046                 getImage() const;
00047
00061             Image::Size getImageSize() const;
00062
00076             Image::Resolution getImageResolution() const;
00077
00091             uint32_t getImageColorDepth() const;
00092
00102             Image::CompressionAlgorithm
00103                 getCompressionAlgorithm() const;
00104
00120             Image::Resolution getScanResolution() const;
00121
00122         protected:
00123             View();
00124             ~View();
00125
00132             void setImageSize(
00133                 const BiometricEvaluation::Image::Size &imageSize);
00134
00141             void setImageColorDepth(uint32_t imageColorDepth);
00142
00149             void setImageResolution(
00150                 const BiometricEvaluation::Image::Resolution
00151                 &imageResolution);
00152
00159             void setScanResolution(
00160                 const BiometricEvaluation::Image::Resolution
00161                 &scanResolution);
00162
00169             void setImageData(
00170                 const BiometricEvaluation::Memory::uint8Array
00171                 &imageData);
00172
00177             void setCompressionAlgorithm(
00178                 const Image::CompressionAlgorithm &ca);
00179
00180         private:
00181             /*
00182              * Items for the Image: Data, resolution, etc.
00183              */
00184             Image::Size _imageSize{};
00185             Image::Resolution _imageResolution{};
00186             Image::Resolution _scanResolution{};
00187             Memory::AutoArray<uint8_t> _imageData;
00188             Image::CompressionAlgorithm
00189                 _compressionAlgorithm{};
00190             uint32_t _imageColorDepth{};
00191
00192         };
00193     }
00194 }
00195 #endif /* __BE_VIEW_VIEW_H__ */
00196
```

Index