BIOMETRIC EVALUATION COMMON FRAMEWORK

PROGRAMMER'S GUIDE
VERSION 0.1

WAYNE SALAMON
GREGORY FIUMARA

IMAGE GROUP
INFORMATION ACCESS DIVISION
INFORMATION TECHNOLOGY LABORATORY

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

NOVEMBER 30, 2017

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 [21].

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++[27] 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) [19] 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 [20]. 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);
    }

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

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

```
#include <be_io_dbrecstore.h>
#include <be_memory_autoarray.h>

#include <iostream>

using namespace BiometricEvaluation;

int
main(
    int argc,
    char *argv[])
{
        IO::DBRecordStore rs("db_recstore", ".", IO::READONLY);

        uint64_t value_size = 0;
        string key("");
        Memory::AutoArray<uint8_t> value;
        for (bool stop = false; stop == false; ) {
                try {
                        // Non-destructively resize the AutoArray to hold
                        // the next record.
                        value.resize(rs.sequence(key, NULL));

                        // Read the record into the AutoArray (treats the
                        // AutoArray as a pointer).
                        rs.read(key, value);

                        // Do something with value.
                        std::cout << "Key " << key << " has a value of " <<
                            value.size() << " bytes" << std::endl;
                } catch (Error::ObjectDoesNotExist) {
                        stop = true;
                }
        }

        return (0);
}
```

`AutoArray` is adapted from ”`c_array` ” [27, 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`

```
#include <be_memory_autoarray.h>
#include <be_memory_indexedbuffer.h>

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

        uint64_t size = IO::Utility::getFileSize("BiometricRecord");
        FILE *fp = std::fopen("BiometricRecord", "rb");
        Memory::IndexedBuffer iBuf(size);
        fread(iBuf, 1, size, fp);
        fclose(fp);
        Memory::IndexedBuffer iBuf(recordData, recordData.size());

        uint32_t lval;
        uint16_t sval;

        /*
         * Record is big-endian:
         * ---------------------------------
         * | NAME | LENGTH | ID |    ...   |
         * ---------------------------------
         *     4       4       2
         */

        /* Read a 4-byte C string */
        lval = iBuf.scanU32Val();        /* Format ID */
        char *cptr = (char *)&lval;
        string s(cptr);

        /* Read a 4-byte length */
        lval = iBuf.scanBeU32Val()

        /* Read a 2-byte ID */
        sval = iBuf.scanBeU16Val();
}
```

Chapter 5

Error Handling

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

5.1 Biometric Evaluation Exceptions

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

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

Listing 6.2 on page 19 shows an example of exception handling when using the logging classes described in Section 6.3 on page 18.

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

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`

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

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

6.3.1 FileLogsheet

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

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

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

Listing 6.2: Using a FileLogsheet within a FileLogCabinet

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

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

```
ls->newEntry();           // Forces the write of the current entry
.........
delete ls;
return;                   // The LogCabinet is destructed by the unique_ptr
```

6.3.2 SysLogsheet

The `SysLogsheet` is an implementation of `Logsheet` which writes log entries to a system logger service. Objects of this class are created with a URL starting with `syslog://`. When using a system logger, the URL must give the hostname of the logger as well as the network port: `syslog://node00:4315` for example. The system logger must understand the Syslog protocol as specified in RFC5424 [28].

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

6.4 Properties

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

Listing 6.3: Using a `Properties` Object

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

6.5 Compressor

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

Listing 6.4: Using a `Compressor` Object

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

Listing 6.5: Setting `Compressor` Options

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

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

Chapter 7

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::Text2`.

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 18), 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. 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 limit 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 timer = new Time::Timer();

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

8.2 Limiting Execution Time

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

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

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

Listing 8.2 shows how an application can use a `Watchdog` object to limit the about 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 timer;

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

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

Any processing that is normally done at the end of the `WATCHDOG` block must also be done within the `expired()` check due to the fact that process control jumps to the end of the `WATCHDOG` block in the event of a timeout. A typical example is the use of the `Timer` object inside a `WATCHDOG` block, as the example in Listing 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 can be used by the application itself to gather statistics related to the current amount of memory being used, the number of threads, and other items. Biometric evaluation test drivers are linked against a third party library, and therefore, the application writer does not control the thread count or memory usage for much of the processing. Listing 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;
    uint64_t userstart, userend;
    uint64_t systemstart, systemend;
    uint64_t diff;
    try {
        stats.getCPUTimes(&userstart, &systemstart);

        // Do some long processing....

        stats.getCPUTimes(&userend, &systemend);
        diff = userend - userstart;
        cout << "User time elapsed is " << diff << endl;
        diff = systemend - systemstart;
        cout << "System time elapsed is " << diff << endl;
    } catch (Error::Exception) {
        cout << "Caught " << e.getInfo() << endl;
    }

}
```

In addition to using the `Process` API to gather statistics to be returned from the function call, the API provides a means to have a "standard" set of statistics logged either synchronously or asynchronously to a `LogSheet` (See Section 6.3 on page 18) contained within a `LogCabinet`. Applications can start and stop logging at will to this `LogSheet`. Post-mortem analysis can then be done on the entries in the log. Listing 9.2 shows the use of logging.

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

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

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

Listing 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[])
{
    IO::LogCabinet lc("statLogCabinet", "Cabinet for Statistics", "");

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

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

    // Do some other work

    // Stop logging
    logstats->stopAutoLogging();
    delete logstats;
}
```

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 parallelized.

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

```
#include <cstdlib>
#include <tr1/memory>
#include <queue>

#include <restaurant.h>

#include <be_process_forkmanager.h>

using namespace std;
using namespace BiometricEvaluation;
using namespace Restaurant;

class ResponsibleEmployeeTask : public Process::Worker
{
public:
        int32_t
        workerMain()
        {
                int32_t status = EXIT_FAILURE;

                /* Retrieve objects assigned to this Task */
                tr1::shared_ptr<Employee> employee =
                    tr1::static_pointer_cast<Employee>(
                    this->getParameter("employee"));
                tr1::shared_ptr< queue<Customer*> > customers =
                    tr1::static_pointer_cast< queue<Customer*> >(
                    this->getParameter("customers")

                employee->clockIn();

                Customer *customer;
                /* Checkpoint after each customer */
                while (this->stopRequested() == false ||
                    employee->isShiftOver() == false) {
                        customer = customers->front();

                        if (customer != NULL) {
                                employee->takeOrder(customer);
                                employee->cookFood(customer);
                                employee->deliverOrder(customer);

                                customers->pop();
                        }
                }

                employee->settleCashDrawer();
                employee->clockOut();

                status = EXIT_SUCCESS;
                return (status);
        }
        ~ResponsibleEmployeeTask() {}
};
```

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

Listing 9.4 is a continuation of Listing 9.3 on page 29 demonstraiting 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).
         */
        while (shiftLeader->getNumActiveWorkers() > 0)
                sleep(1);

        shiftLeader->armAlarmAndExit();

        status = EXIT_SUCCESS;
        return (status);
}
```

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 29 and Listing 9.4 on the preceding page respectively, showing an example of communication, using `std::string` messages.

Listing 9.5: `Worker` Communication

```
        Memory::uint8Array msg;

        /* Deal with next customer unless Manager interrupts in next second */
        if (this->waitForMessage(1)) {
                if (this->receiveMessageFromManager(msg)) {
                        Action action = Restaurant::messageToAction(msg);
                        switch (action) {
                        case TAKE_BREAK:
                                employee->goOnBreak();
                                break;
                        /* ... */
                        }
                }
        }

        /* ... */

        if (customer->isComplaining()) {
                sprintf((char *)&(*msg), "Customer Complant");
                this->sendMessageToManager(msg);
        }
```

Listing 9.6: `Manager` Communication

```
        tr1::shared_ptr<Process::WorkerController> sender;
        Memory::uint8Array msg;

        /* Do routine tasks unless employee has concern in the next 2 seconds */
        while (this->getNextMessage(sender, msg, 2)) {
                Action action = Restaurant::messageToAction(msg);
                switch (action) {
```

```
                case CUSTOMER_COMPLAINT:
                        sprintf((char *)&(*msg), "I'll take care of it.");
                        this->sendMessageToWorker(msg);
                        break;
                /* ... */
                }
        }

        /* ... */

        /*  Closing Time */
        sprintf((char *)&(*msg), "Clock out and go home.");
        this->broadcastMessage(msg);
```

Chapter 10

System

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

Listing 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();
        Process::Statistics::getMemorySizes(NULL, &vmSize, NULL, NULL, NULL);
        memSize -= vmSize;    // subtract off memory used by parent

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

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

Chapter 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 biometic records have fields for image attributes like dimensions and resolution, the corresponding fields of an `Image` class are **not** populated with their contents. The `Image` namespace data types *are* used outside of the namespace, such as in finger views, to retrieve image attributes stored as part of the biometric record. Applications can compare those values against the values within the `Image` object, as in most cases those values are taken directly from the underlying image data. See Chapter 15 on page 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 [15]. Decompression and grayscale conversion are accomplished via `libjpeg` [13].

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` [20].

11.6 JPEG2000

The `JPEG2000` class provides Image class functionality to JPEG 2000-encoded images [14]. 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`) [18]. `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 [10]. Decompression is provided by `libpng` [24].

`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` [25]. 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 [29] are represented by the `WSQ` class. The WSQ decompressor found in NIST Biometric Image Software [20], `libwsq`, is used by this class. The class provides a static function to determine whether or not an image appears to be encoded in the WSQ format.

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

Chapter 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 [11] 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

```
#include <iostream>
#include <be_memory_autoarray.h>
#include <be_io_utility.h
#include <be_video_container.h>

```

```
using namespace BiometricEvaluation;
using namespace std;

int
main(int argc, char* argv[])
{
        std::unique_ptr<Video::Container> pvc;

        std::string filename = "./test_data/2video1audio.mp4";
        if ((argc != 1) && (argc != 2)) {
                cerr << "usage: " << argv[0] << " [filename]" << endl
                    << "If <filename> is not given, " << filename
                    << " is used instead." << endl;
                return (EXIT_FAILURE);
        }
        if (argc == 2)
                filename = argv[1];

        cout << "Construct an program stream from file "
            << filename << endl;
        /*
         * Three ways to open the container:
         * 1) Have the framework open the file directly;
         * 2) Read the file into a local buffer and give that to the framework;
         * 3) Read the file into a buffer wrapped in a shared pointer and pass
         *    that to the framework.
         */
        try {
//              pvc.reset(new
//                  Video::Container(filename));

//              Memory::uint8Array buf =
//                  IO::Utility::readFile(filename);
//              pvc.reset(new Video::Container(buf));

                std::shared_ptr<Memory::uint8Array> buf;
                buf.reset(new Memory::uint8Array(
                    IO::Utility::readFile(filename)));
                pvc.reset(new Video::Container(buf));
        } catch (Error::Exception &e) {
                cout << "Caught: " << e.whatString() << endl;
                return (EXIT_FAILURE);
        }

        cout << "Video Count: " << pvc->getVideoCount() << endl;

        std::unique_ptr<Video::Stream> stream;
        /*
         * Open the first video stream.
         */
        try {
                stream = pvc->getVideoStream(1);
        } catch (Error::Exception &e) {
                cerr << "Could not retrieve video stream: " << e.whatString()
                    << endl;
                return (EXIT_FAILURE);
        }
        /*
         * Read all the frames, one at a time, scaled down and converted
         * to 8-bit grayscale.
         */
        float scaleFactor = 0.5;
        Image::PixelFormat pixelFormat = Image::PixelFormat::Gray8;
        stream->setFrameScale(scaleFactor, scaleFactor);
        stream->setFramePixelFormat(pixelFormat);
        uint64_t expectedCount = stream->getFrameCount();

        cout << "First video stream: " << stream->getFPS() << " FPS, "
            << expectedCount << " frames." << endl;
        /*
         * The frame count can be zero, meaning unknown. If that is the case,
         * loop until a parameter error is indicated.
         */
        if (expectedCount == 0)
                expectedCount = 99999999;
        uint64_t count = 0;
        for (uint64_t f = 1; f <= expectedCount; f++) {
                try {
                        auto frame = stream->getFrame(f);
                        count++;
                        /* Do something with frame.data */
                        std::cout << "frame size is "
                            << frame.size.xSize << "x" << frame.size.ySize
                            << std::endl;
                } catch (Error::ParameterError &e) {
                        cout << "No more frames.";
                        break;
                } catch (Error::Exception &e) {
                        std::cout << "Caught " << e.whatString() << endl;
                        return (EXIT_FAILURE);
                }
        }
        cout << "Retrieved " << count << " frames." << endl;
        return (EXIT_SUCCESS);
}
```

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

```
#include <iostream>
#include <be_data_interchange_an2k.h>
#include <be_feature_an2k11efs.h>

```

```
/*
 * This test program exercises the Evaluation framework to process AN2K
 * records stored in a RecordStore. The intent is to model what a real
 * program would do by retrieving AN2K records, doing some processing
 * on the image, and displaying the results.
 */
using namespace BiometricEvaluation;

static void
printAN2K11EFS(Feature::AN2K11EFS::ExtendedFeatureSet &efs)
{
	Image::ROI roi = efs.getImageInfo().roi;
	std::cout << "ROI:\n"
	    << "\tSize: ("
	    << roi.size.xSize << "," << roi.size.ySize << ")\n"
	    << "\tOffset: ("
	    << roi.horzOffset << "," << roi.vertOffset << ")\n"
	    << "\tPath: ";
	for (auto const& point: roi.path) {
		std::cout << point << " ";
	}
	std::cout << "\n";

	std::cout << "Image Info:\n" << efs.getImageInfo() << "\n\n";

	Feature::AN2K11EFS::CorePointSet cps = efs.getCPS();
	std::cout << "CPS: Have " << cps.size() << " EFS core point(s):\n";
	for (auto const& cp: cps) {
		std::cout << "\t" << cp << "\n";
	}

	Feature::AN2K11EFS::DeltaPointSet dps = efs.getDPS();
	std::cout << "DPS: Have " << dps.size() << " EFS delta point(s):\n";
	for (auto const& dp: dps) {
		std::cout << "\t" << dp << "\n";
	}

	Feature::AN2K11EFS::MinutiaPointSet mps = efs.getMPS();
	std::cout << "MPS: Have " << mps.size() << " EFS minutia point(s):\n";
	for (auto const& mp: mps) {
		std::cout << mp << "\n";
	}

	std::cout << "No Features Present:\n";
	std::cout << efs.getNFP();

	std::cout << "\nMinutiae Ridge Count Information:\n";
	auto mrci = efs.getMRCI();
	std::cout << mrci << "\n";
}

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

	std::string fname = "test_data/type9-efs.an2k";
	/*
```

```
	 * Read the EFS data from the DataInterchange::AN2KRecord object
	 */
	std::cout << "Extended Feature Set data in " << fname << ": ";
	try {
		DataInterchange::AN2KRecord an2k(fname);
		std::vector<Finger::AN2KMinutiaeDataRecord> minutiae =
		    an2k.getMinutiaeDataRecordSet();
		printAN2K11EFS(*minutiae[0].getAN2K11EFS());
	} catch (Error::Exception &e) {
		std::cout << "Failed; caught " << e.whatString() << "\n";
	}

	/*
	 * Read the EFS data by constructing directly from the filename
	 */
	try {
		Feature::AN2K11EFS::ExtendedFeatureSet efs(fname, 1);
		printAN2K11EFS(efs);
	} catch (Error::Exception &e) {
		std::cout << "Failed; caught " << e.whatString() << "\n";
	}
}
```

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.

`View` s are high-level abstractions of the biometric sample, and concrete implementations of a `View` include finger, face, iris, etc. views based on a specific type of biometric. Therefore, `View` objects are not created directly, Subclasses, such as finger views (see Chapter 16 on page 53), represent the specific type of biometric sample.

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

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

Listing 15.1: View::View Class

```
void
printViewInfo(BiometricEvaluation::View::View &view)
{
        cout << "Image size is " << view.getImageSize() << endl;
        cout << "Image resolution is " << view.getImageResolution() << endl;
        cout << "Scan resolution is " << view.getScanResolution() << endl;
        cout << "Image color depth is " << view.getImageColorDepth() << endl;
        cout << "Compression is " << view.getCompressionAlgorithm() << endl;
        try {
                auto theImage = view.getImage();
                cout << "Information from the Image data item:" << endl;
                cout << "\tResolution: " << theImage->getResolution() << endl;
                cout << "\tDimensions: " << theImage->getDimensions() << endl;
                cout << "\tDepth: " << theImage->getColorDepth() << endl;
        } catch (Error::Exception &e) {
                cout << "Caught " << e.what() << endl;
        }
}
```

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;
        } else {
                // Process the image data
        }
        /*
         * Get the raw image data and save to a file
         */
        std::ofstream img_out("imgdata.raw", std::ofstream::binary);
        Memory::uint8Array imgData{img->getRawData()};
        img_out.write((char *)&(imgData[0]), imgData.size());
        if (img_out.good()) {
                img_out.close();
        } else {
                std::cout << "Error occurred when writing." << std::endl;
        }
        /*
         * Get all the positions from the data record.
         */
        Finger::PositionSet positions = an2kv->getPositions();
        std::cout << "There are " << positions.size() << " positions:"
            << std::endl;
        for (auto p: positions) {
                std::cout << "\t" << to_string(p) << std::endl;
        }
        /*
         * Get the minutiae data records and print the minutiae points in
         * each data record
         */
        auto mdrs = an2kv->getMinutiaeDataRecordSet();  // The set of records
        std::cout << "There are " << mdrs.size() << " minutiae data records."
            << std::endl;
        for (auto mdr: mdrs) {
                for (auto mp: mdr.getAN2K7Minutiae()->getMinutiaPoints()) {
                        std::cout << mp << std::endl;
                }
        }

        return(EXIT_SUCCESS);
}
```

16.1.2 ISO/INCITS Finger Views

The ISO [17] and INCITS [16] 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

```
#include <iostream>
#include <be_finger_ansi2004view.h>
#include <be_feature_incitsminutiae.h>
```

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

int
main(int argc, char* argv[])
{
    Finger::ANSI2004View fngv;
    try {
        fngv = Finger::ANSI2004View("test_data/fmr.ansi2004", "", 3);
    } catch (Error::Exception &e) {
        cerr << "Caught " << e.whatString()  << endl;
        return (EXIT_FAILURE);
    }
    cout << "Image resolution is " << fngv.getImageResolution() << endl;
    cout << "Image size is " << fngv.getImageSize() << endl;
    cout << "Image color depth is " << fngv.getImageColorDepth() << endl;
    cout << "Compression is " << fngv.getCompressionAlgorithm() << endl;
    cout << "Scan resolution is " << fngv.getScanResolution() << endl;

    Feature::INCITSMinutiae fmd = fngv.getMinutiaeData();
    cout << "Minutiae format is " << fmd.getFormat() << endl;
    Feature::MinutiaPointSet mps = fmd.getMinutiaPoints();
    cout << "There are " << mps.size() << " minutiae points:" << endl;
    for (auto mp: mps)
        cout << mp;

    Feature::RidgeCountItemSet rcis = fmd.getRidgeCountItems();
    cout << "There are " << rcis.size() << " ridge count items:" << endl;
    for (auto rci: rcis)
        cout << "\t" << rci;

    Feature::CorePointSet cores = fmd.getCores();
    cout << "There are " << cores.size() << " cores:" << endl;
    for (auto core: cores)
        cout << "\t" << core;

    Feature::DeltaPointSet deltas = fmd.getDeltas();
    cout << "There are " << deltas.size() << " deltas:" << endl;
    for (auto delta: deltas)
        cout << "\t" << delta;

    exit (EXIT_SUCCESS);
}
```

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.

17.0.1 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

```cpp
#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) {
		cout << "Image info:" << endl;
```

```
                cout << "\tCompression: " << img->getCompressionAlgorithm()
                    << endl;
                cout << "\tDimensions: " << img->getDimensions() << endl;
                cout << "\tResolution: " << img->getResolution() << endl;
                cout << "\tDepth: " << img->getColorDepth() << endl;
        } else {
                cout << "No Image available." << endl;
        }

}

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

        /*
         * Call the constructor that will open an existing AN2K file.
         */
        std::shared_ptr<Palm::AN2KView> an2kv;
        try {
                an2kv.reset(new Palm::AN2KView(
                    "test_data/type9-15.an2k", 1));
        } catch (Error::Exception &e) {
                cout << "Caught " << e.what()  << endl;
                return (EXIT_FAILURE);
        }
        printViewInfo(*an2kv);

        cout << "Get the set of minutiae data records: ";
        auto minutiae = an2kv->getMinutiaeDataRecordSet();
        cout << "There are " << minutiae.size()
            << " minutiae data record sets." << endl;
        if (minutiae.size() != 0) {
                cout << "Minutiae Points:\n";
                for (auto m:
                     minutiae[0].getAN2K7Minutiae()->getMinutiaPoints()) {
                        cout << m << endl;
                }
                cout << "Cores:\n";
                for (auto c:
                     minutiae[0].getAN2K7Minutiae()->getCores()) {
                        cout << c << endl;
                }
                cout << "Deltas:\n";
                for (auto d:
                     minutiae[0].getAN2K7Minutiae()->getDeltas()) {
                        cout << d << endl;
                }
        }
        return(EXIT_SUCCESS);
}
```

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;
                cout << "\tDimensions: " << theImage->getDimensions() << endl;
                cout << "\tDepth: " << theImage->getColorDepth() << endl;
        } catch (Error::Exception &e) {
                cout << "Caught " << e.what() << endl;
        }
        cout << "-----------------------------------------" << endl;
}

void
printFaceInfo(Face::ISO2005View &facev)
{
        /*
         * Provided by the Face::INCITSView interface.
         */
        cout << "Gender: " << facev.getGender() << endl;
        cout << "Eye Color: " << facev.getEyeColor() << endl;
        cout << "Hair Color: " << facev.getHairColor() << endl;
        cout << "Expression: " << facev.getExpression() << endl;

        Face::PoseAngle pa =  facev.getPoseAngle();
        cout << "Pose angle info: ";
        cout << "Yaw/Uncer: " << (int)pa.yaw << "/" << (int)pa.yawUncertainty;
        cout << "; Pitch/Uncer: "
            << (int)pa.pitch << "/" << (int)pa.pitchUncertainty;
        cout << "; Roll/Uncer: "
            << (int)pa.roll << "/" << (int)pa.rollUncertainty << endl;

        cout << "Image type is " << facev.getImageType() << endl;
        cout << "Image data type is " << facev.getImageDataType()
            << endl;
        cout << "Color space is " << facev.getColorSpace() << endl;
        cout << "Source type is " << facev.getSourceType() << endl;
        cout << "Device type is " << "0x" << hex << setw(4) << setfill('0')
            << facev.getDeviceType() << dec << endl;

        Face::PropertySet properties;
        bool haveProps = facev.propertiesConsidered();
        if (haveProps) {
                facev.getPropertySet(properties);
                cout << "There are " << properties.size() << " properties: ";
                for (size_t i = 0; i < properties.size(); i++) {
                        if (i != properties.size() - 1)
                                cout << properties[i] << ", ";
                        else
                                cout << properties[i];
                }
                cout << endl;
        } else {
                cout << "There are no properties." << endl;
        }

        Feature::MPEGFacePointSet fps;
        facev.getFeaturePointSet(fps);
        cout << "There are " << fps.size() << " feature points." << endl;
        if (fps.size() != 0) {
                cout << "\tType\tCode\tPosition" << endl;
        }
        for (size_t i = 0; i < fps.size(); i++) {
                cout << "\t" << (int)fps[i].type
                    << "\t" << (int)fps[i].major << "." << (int)fps[i].minor
                    << "\t" << fps[i].coordinate
                    << endl;
        }
        cout << "-------------------------------------------" << endl;
}

int
main(int argc, char* argv[])
{
        Face::ISO2005View facev;
        try {
                facev = Face::ISO2005View("test_data/face01.iso2005", 1);
        } catch (Error::Exception &e) {
                cout << "Caught " << e.what()  << endl;
                return (EXIT_FAILURE);
        }
        printViewInfo(facev);
        printFaceInfo(facev);
        return(EXIT_SUCCESS);
}
```

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

```
#include <iostream>
#include <be_data_interchange_an2k.h>

/*
 * This test program exercises the Evaluation framework to process an AN2K
 * records stored in a file. The intent is to model what a real program
 * 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.
         */
        auto minutiae = an2kv.getMinutiaeDataRecordSet();
        cout << "\tThere are " << minutiae.size() <<
            " minutiae data records.\n";
}

int
main(int argc, char* argv[]) {
        try {
                DataInterchange::AN2KRecord an2k("test_data/a002.an2");
                printRecordInfo(an2k);
                /*
                 * Obtain the finger capture and latent views from the
                 * AN2k file.
                 */
                int i = 0;
                for (auto c: an2k.getFingerCaptures()) {
                        cout << "[Capture View " << i++ <<"]\n";
                        printViewInfo(c);
                        cout << "\tPosition: " << c.getPosition()
                            << endl;
                        cout << "[End of Capture View]\n";
                }
                i = 0;
                for (auto l: an2k.getFingerLatents()) {
                        cout << "[Latent View " << i++ <<"]\n";
                        printViewInfo(l);
                        cout << "\tPositions: ";
                        for (auto p: l.getPositions()) {
                                cout << p << " ";
                        }
                        cout << endl << "[End of Latent View]\n";
                }
                /*
                 * Obtain the entire set of minutiae records from the
                 * AN2k file, independently of the view.
                 */
                auto minutiae = an2k.getMinutiaeDataRecordSet();
                cout << "There is a total of " << minutiae.size()
                    << " minutiae data records in the AN2K file.\n";
                cout << ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n";
        } catch (Error::Exception &e) {
                cout << "Failed sequence: " << e.what() << endl;
                return (EXIT_FAILURE);
        }
}
```

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 [16]. 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);
}

```

```
int
main(int argc, char* argv[])
{
        std::unique_ptr<DataInterchange::ANSI2004Record> record;

        /* Construct with a file, minutia record only. */
        try {
                record.reset(new DataInterchange::ANSI2004Record(
                    "test_data/fmr.ansi2004", ""));
        } catch (Error::Exception& e) {
                cout << "A file error occurred: " << e.what() << endl;
                return (EXIT_FAILURE);
        }

        /* Remove all views but the first */
        record->isolateView(1);
        showAllViews(*record);

        /* Modify the minutia in a finger view */
        auto minutiaRecord = record->getMinutia(1);
        auto minutiaPoints = minutiaRecord.getMinutiaPoints();
        for (auto& fm: minutiaPoints) {
                fm.coordinate.x += 10;
                fm.coordinate.y += 10;
        }
        /* Replace minutiae in the remaining view */
        minutiaRecord.setMinutiaPoints(minutiaPoints);
        record->setMinutia(1, minutiaRecord);
        showAllViews(*record);

        /* Obtain the ANSI-378 record and instantiate an object from it */
        auto fmr = record->getFMR();
        BE::Finger::ANSI2004View fmrView(fmr, Memory::uint8Array{}, 1);
        /* The fmr object can also be written to a file */

        return (EXIT_SUCCESS);
}
```

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 apppli-cation 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 32).

Listing 20.1: Basic `MessageCenter` Usage

```
namespace BE = BiometricEvaluation;

uint32_t clientID;
BE::Memory::uint8Array message;
BE::Process::MessageCenter mc;
for (;;) {
        /* ... do work ... */

        if (mc.hasUnseenMessages()) {
                mc.getNextMessage(clientID, message);
                std::cout << clientID << " sent a " << message.size() <<
                    " byte message." << std::endl;

                Memory::AutoArrayUtility::setString(message, "ACK\n");
                mc.sendResponse(clientID, message);
        }
}
```

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

```
namespace BE = BiometricEvaluation;

enum class TestCommand
{
        Stop,
        Help
};

template<>
const std::map<TestCommand, std::string>
BE::Framework::EnumerationFunctions<TestCommand>::enumToStringMap {
        {TestCommand::Stop, "STOP"},
        {TestCOmmand::Help, "HELP"}
};

class TestCommandParser : public BE::Process::CommandParser<TestCommand>
{
public:
        void
        parse(
            const BE::Process::CommandParser<TestCommand>::Command &command)
        {
                switch (command.command) {
                case TestCommand::Stop:
                        this->stop(command);
                        break;
                case TestCommand::Help:
                        this->help(command);
                        break;
                }
        }

private:
        void
        stop(
            const BE::Process::CommandParser<TestCommand>::Command &command)
        {
                /* Ensure proper arguments */
                if (command.arguments.size() != 1) {
                        this->sendResponse(command.clientID, "Usage: " +
                            to_string(command.command) + " <process>");
                        return;
                }

                /* ... perform stop operation ... */
        }

        void
        help(
            const BE::Process::CommandParser<TestCommand>::Command &command)
        {
                this->sendResponse(command.clientID, "Available Commands:\n"
                    "\tSTOP <process>\n\tHELP");
        }
};

int
main()
{
        TestCommandParser commandCenter;
        TestCommandParser::Command command;
        for (;;) {
                /* ... do work ... */

                if (commandCenter.hasPendingCommands()) {
                        commandCenter.getNextCommand(command);
                        commandCenter.parse();
                }
        }

        return (EXIT_SUCCESS);
}
```

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

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

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

Second, each Task-N can perform some work before creating the workers. One example is the loading of a large data set into memory; again, this is done within the application. Because Task-N calls the POSIX function `fork()` to create the workers, each worker inherits the work done by Task-N. In the case of a memory load, each worker now has that memory mapped into it's address space. See Section 21.5 on page 75 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.

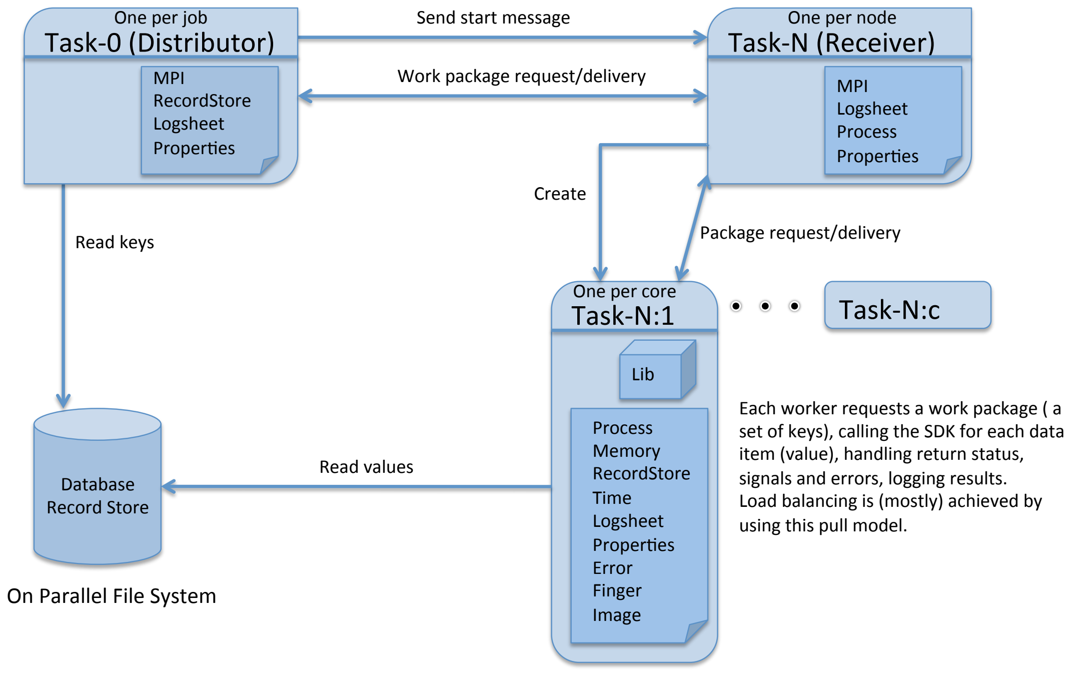


Figure 21.1: MPI Parallel Job Processes and Data Flow

The classes `RecordStoreDistributor` (Section 21.3.1) and `RecordProcessor` (Section 21.5.1 on the following page) are examples of `WorkPackage` clients that insert and remove data from a work package.

21.3 Distributor

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

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

21.3.1 Record Store Distributor

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

An application can create an instance of a `RecordStoreDistributor` with the name of a record store in order to distribute records for processing across the MPI job. Listing 21.3 on page 82 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.

21.4 Receiver

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

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

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

21.5 Work Package Processor

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

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

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

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

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

21.5.1 Record Processor

`RecordProcessor` is a partial implementation of `WorkPackageProcessor` and defines the `processWorkPackage()` of the `WorkPackageProcessor` interface; other methods are declared as pure-virtual and must be implemented by a child class. In addition, `RecordProcessor` declares a new pure-virtual method, `processRecord()` to be implemented by a subclass to process a single record from the record store. In summary, `RecordProcessor` removes records from the work package to be processed within the subclass, which is defined by the application. See Listing 21.1 on page 78 and Listing 21.2 on page 79 for a example of such an implementation.

21.6 MPI Resources

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

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

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

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

The `Logsheet URL` property is optional, and if present all MPI Framework trace messages will be written to the specified logging target. Two types of Uniform Resource Locator schemes are allowed: `file://` and `syslog://`, corresponding to the types of `Logsheet` classes (Section 6.3 on page 18) in the Framework.

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

Input Record Store The input record store;

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

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

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

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

21.7 MPI Runtime

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

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

SIGQUIT Exit when the current work package is exhausted;

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

SIGTERM Exit immediately ("termination exit").

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

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

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.

21.8 Logging

In order to aid tracing and debugging of a parallel job, the MPI Framework can be configured to write trace messages to the log storage. These trace messages are logged as debug messages instead of normal entries. The type and location of the log is given to the Framework by using a URL as a property when starting the MPI job (see Section 21.6 on the facing page).

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

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

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

21.9 MPI Framework Applications

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

Listing 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

```
class TestRecordProcessor : public BiometricEvaluation::MPI::RecordProcessor {
public:
        /**
         * @brief
         * The property string ''Logsheet URL''.
         */
        static const std::string RECORDLOGSHEETURLPROPERTY;

        static const uint32_t SHAREDMEMORYSIZE = 2048;

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

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

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

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

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

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

Next, Listing 21.2 on the facing page 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

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

#include "test_be_mpi.h"

using namespace BiometricEvaluation;

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

/*
 * Implementations of the MPI RecordProcessor class interface.
 * Calls the parent constructor to manage the properties file name.
 */
TestRecordProcessor::TestRecordProcessor(
    const std::string &propertiesFileName) :
    RecordProcessor(propertiesFileName)
{
}

TestRecordProcessor::~TestRecordProcessor()
{
}

/*
 * Factory object: Log our call and set up the shared memory buffer.
 */
void
TestRecordProcessor::performInitialization(
    std::shared_ptr<IO::Logsheet> &logsheet)
{
        this->setLogsheet(logsheet);

        /*
         * Set up the memory that will be shared across all instances.
         */
        char *buf = (char *)malloc(SHAREDMEMORYSIZE);
        strcpy(buf, "SHARED MEMORY");
        this->_sharedMemorySize = SHAREDMEMORYSIZE;
        this->_sharedMemory = std::unique_ptr<char>(buf);

        *logsheet.get() << std::string(__FUNCTION__) << " called: ";
        *logsheet.get()
            << "Shared memory size is " << this->_sharedMemorySize
            << " and contents is [" << buf << "]";
        BE::MPI::logEntry(*logsheet.get());
}

/*
 * Factory object: Create a new instance of the TestRecordProcess
 * that will work on work package records. Each instance gets
 * its own instance of the log sheet.
 */
std::shared_ptr<BiometricEvaluation::MPI::WorkPackageProcessor>
```

```
TestRecordProcessor::newProcessor(
    std::shared_ptr<IO::Logsheet> &logsheet)
{
        std::string propertiesFileName =
            this->getResources()->getPropertiesFileName();
        TestRecordProcessor *processor =
            new TestRecordProcessor(propertiesFileName);
        processor->setLogsheet(logsheet);

        /*
         * If we have our own Logsheet property, and we can open
         * that Logsheet, use it for record logging; otherwise,
         * create a Null Logsheet for these events. We use the
         * framework's Logsheet for tracing of processing, not
         * record handling logs.
         */
        std::string url;
        std::unique_ptr<BE::IO::PropertiesFile> props;
        try {
                /* It is crucial that the Properties file be
                 * opened read-only, else it will be rewritten
                 * when the unique ptr is destroyed, causing
                 * a race condition with other processes that
                 * are reading the file.
                 */
                props.reset(new BE::IO::PropertiesFile(
                    propertiesFileName, IO::READONLY));
                url = props->getProperty(
                    TestRecordProcessor::RECORDLOGSHEETURLPROPERTY);
        } catch (BE::Error::Exception &e) {
                url = "";
        }
        processor->_recordLogsheet = BE::MPI::openLogsheet(
            url, "Test Record Processing");
        processor->_sharedMemory = this->_sharedMemory;
        processor->_sharedMemorySize = this->_sharedMemorySize;

        std::shared_ptr<BiometricEvaluation::MPI::WorkPackageProcessor> sptr;
        sptr.reset(processor);
        return (sptr);
}

/*
 * Helper function to log some information about a record.
 */
static void
dumpRecord(
    BE::IO::Logsheet &log,
    const std::string key,
    const Memory::uint8Array &val)
{
        log << "Key [" << key << "]: ";
        /* Dump some bytes from the record */
        for (uint64_t i = 0; i < 8; i++) {
                log << std::hex << (int)val[i] << " ";
        }
        log << " |";
        for (uint64_t i = 0; i < 8; i++) {
                log << (char)val[i];
        }
        log << "|";
        BE::MPI::logEntry(log);
}

/*
 * The worker object: Log to the Framework Logsheet, obtain the data for
 * the record, and log some information to the record Logsheet.
 */
void
TestRecordProcessor::processRecord(const std::string &key)
{
        BE::IO::Logsheet *log = this->getLogsheet().get();

        if (this->getResources()->haveRecordStore() == false) {
                BE::MPI::logMessage(*log, "processRecord(" + key + ")"
                    + " called but have no record store; returning.");
                return;
        }
        *log << "processRecord(" << key << ") called: ";
        char *buf = this->_sharedMemory.get();
        *log << "Shared memory size is " << this->_sharedMemorySize
            << " and contents is [" << buf << "]";
        BE::MPI::logEntry(*log);

        Memory::uint8Array value(0);
        std::shared_ptr<IO::RecordStore> inputRS =
            this->getResources()->getRecordStore();
        try {
                inputRS->read(key, value);
        } catch (Error::Exception &e) {
                *log << string(__FUNCTION__) <<
                    " could not read record: " <<
                    e.whatString();
                return;
        }
        /*
         * Log record info to our own Logsheet instead of
         * the one provided by the framework.
         */
        BE::IO::Logsheet *rlog = this->_recordLogsheet.get();
        dumpRecord(*rlog, key, value);
}

/*
 * The worker object: Log to the Framework Logsheet, and log some record
 * information to the record Logsheet.
 */
void
TestRecordProcessor::processRecord(
    const std::string &key,
    const BiometricEvaluation::Memory::uint8Array &value)
{
        BE::IO::Logsheet *log = this->getLogsheet().get();
        *log << "processRecord(" << key << ", [value]) called: ";
        char *buf = this->_sharedMemory.get();
        *log << "Shared memory size is " << this->_sharedMemorySize
            << " and contents is [" << buf << "]";
        BE::MPI::logEntry(*log);

        /*
         * Log record info to our own Logsheet instead of
         * the one provided by the framework.
         */
        BE::IO::Logsheet *rlog = this->_recordLogsheet.get();
        dumpRecord(*rlog, key, value);
}
```

Listing 21.3: MPI Framework Application Main

```
int
main(int argc, char* argv[])
{
        /*
         * It is important that the MPI runtime environment be started
         * prior to any other activity that may result in premature
         * termination. Therefore, participate in the MPI environment, but
         * don't create a Receiver or Distributor until any local items
         * are take care of.
         */
        MPI::Runtime runtime(argc, argv);

        std::unique_ptr<MPI::RecordStoreDistributor> distributor;
        std::unique_ptr<MPI::Receiver> receiver;
        std::shared_ptr<TestRecordProcessor> processor;

        /*
         * If there is any argument to the program, use keys only as the
         * record distribution method. Otherwise, use keys and values.
         */
        bool includeValues;
        if (argc == 1) {
                MPI::printStatus("Test Distributor and Receiver, keys only");
                includeValues = false;
        } else {
                MPI::printStatus("Test Distributor and Receiver, keys and values");
                includeValues = true;
        }
        try {
                distributor.reset(
                    new MPI::RecordStoreDistributor(propFile, includeValues));

                processor.reset(new TestRecordProcessor(propFile));

                receiver.reset(new MPI::Receiver(propFile, processor));

                runtime.start(*distributor, *receiver);
                runtime.shutdown();
        } catch (Error::Exception &e) {
                MPI::printStatus("Caught: " + e.whatString());
                runtime.abort(EXIT_FAILURE);
        } catch (...) {
                MPI::printStatus("Caught some other exception");
                runtime.abort(EXIT_FAILURE);
        }

        return (EXIT_SUCCESS);
}
```

References

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

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

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

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

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

[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.

[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.

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

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

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

[11] FFmpeg Multimedia Framework. `http://www.ffmpeg.org`.

[12] GNU Make Project. `http://www.gnu.org/software/make`.

[13] Independent JPEG Group. libjpeg, 2011. `http://www.ijg.org/`.

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

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

[16] International Committee for Information Technology Standards. `http://www.incits.org`.

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

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

[19] Message Passing Interface Forum. `http://www.mpi-forum.org`.

[20] NIST Biometric Image Software, 2011. https://www.nist.gov/services-resources/software/nist-biometric-image-software-nbis.

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

[22] The Open MPI Project. http://www.openmpi.org.

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

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

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

[26] The SQLite Project. https://sqlite.org.

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

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

[29] 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.

Appendix A

Building the Framework

A.1 Language Features

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

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

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

A.2 The Framework Build System

The distribution of BECommon includes a set of `make` files used to build the BECommon library, as well as install the library and header files. These `make` file use some features of the GNU make [12] system, and therefore the GNU software must be installed on the user's system. Future versions of BECommon may use a different build system.

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

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

A.3 External Software Dependencies

The Biometric Evaluation Framework is built upon several other software packages. The packages are used for image processing, biometric data record formats, the message passing interface [19], as well as operating system and compiler tool chains.

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

A.3.1 NIST Biometric Image Software

The NIST Biometric Image Software (NBIS) [20] is a set of packages used for ANSI-NIST [6], WSQ [29] 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.3.2 Video and Image Processing

For the `Image` classes, the JPEG [13], NBIS [20], OpenJPEG [18], PNG [24], and TIFF [25] development libraries are required.

For `Video` classes, the FFmpeg [11] libraries are used. When building from source, configure to build and install shared libraries. By default, only static libraries are built.

A.3.3 Cryptography

Cryptography support is provided by the OpenSSL [23] library. An example is the `openssl-devel` package on Linux systems which provides the `libcrypto` file and associated header files for development.

A.3.4 Sqlite

SQLite is an embedded Structured Query Language (SQL) database engine and is used by the `IO::SQLiteRecordStore` class to provide an `IO::RecordStore` that is backed by a SQLite database. Information on SQLite can be found at [26].

A.3.5 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.3.6 Message Passing Interface

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

One common implementation of MPI is OpenMPI [22], available as source code, or binary packages. Often the MPI runtime is a separate binary package from the MPI development software. As an example, for many Linux distributions, an example of the runtime package is `openmpi-1.6.4-3`, while the related development package would be `openmpi-devel-1.6.4-3`.

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

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

Appendix B

Running an MPI Job

B.1 OpenMPI

This chapter describes how to use the OpenMPI [22] runtime system to execute an MPI job. Some parameters passed to the `mpirun` command are related to the notions captured in the Biometric Evaluation Framework MPI support.

B.2 Example Shell Script

Listing B.1: Example Script to run MPI

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

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

```
EOF

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

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

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

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

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

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

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

Appendix C

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

Namespace Documentation

F.1 BiometricEvaluation Namespace Reference

Namespaces

- **Error**

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

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

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

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

  Information about the framework.
- **Image**

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

  Input/Output functionality.
- **Iris**

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

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

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

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

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

  ***Process** (p. 148) information and controls.*
- **System**

  Operating system, hardware, etc.

- **Text**

  Text (p. *151*) *processing for string objects.*
- **Time**

  Support for time and timers.
- **Video**

  Basic information relating to video and streams.
- **View**

  View (p. *581*) *information.*

F.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. 143) Receiver.

F.2 BiometricEvaluation::Error Namespace Reference

Exceptions, and other error handling.

Classes

- class **ConversionError**

  Error (p. *106*) *when converting one object into another, a property value from string to int, for example.*
- class **DataError**

  Error (p. *106*) *when reading data from an external source.*
- class **Exception**

  The parent class of all ***BiometricEvaluation*** (p. *105*) *exceptions.*
- class **FileError**

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

  An error occurred when allocating an object.
- class **NotImplemented**

  A ***NotImplemented*** (p. *452*) *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. 539) object is used to handle signals that come from the operating system.*
- class **StrategyError**

  *A **StrategyError** (p. 560) 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)

F.2.1 Detailed Description

Exceptions, and other error handling.

The **Error** (p. 106) package contains classes for exceptions, and functions used for error handling, including signals generated by a process.

F.2.2 Function Documentation

F.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.

F.3 BiometricEvaluation::Face Namespace Reference

Biometric information relating to face images and derived information.

Classes

- class **INCITSView**

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

- class **ISO2005View**

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

    Representation of pose angle and uncertainty.

Typedefs

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

Enumerations

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

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

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

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

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

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

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

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

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

    Source type codes.

F.3.1 Detailed Description

Biometric information relating to face images and derived information.

The **Face** (p. 107) 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.

F.3.2 Typedef Documentation

F.3.2.1 PropertySet

```
typedef std::vector< BiometricEvaluation::Face::Property>  BiometricEvaluation::Face::Property↩
Set
```

A set of properties.

F.4 BiometricEvaluation::Feature Namespace Reference

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

Namespaces

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

Enumerations

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

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

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

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

  Enumerate the types of extraction methods for ridge counts.

Functions

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

  Output stream overload for ***FrictionRidgeGeneralizedPosition*** *(p. 342).*
- 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** &)

F.4.1 Detailed Description

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

Definition of an MPEG4 **Face** (p. 107) feature point. See ISO/IEC 14496-2.

F.4.2 Function Documentation

F.4.2.1 operator<<()

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

Output stream overload for **FrictionRidgeGeneralizedPosition** (p. 342).

Parameters

in	*s*	Stream on which to append formatted information.
in	*fgp*	**FrictionRidgeGeneralizedPosition** (p. 342) information to append to stream.

Returns

stream with a fgp textual representation appended.

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

Classes

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

  Sort *(p. 111) by increasing distance from center and angle (theta).*
- class **Quality**
- class **XY**
- class **YX**

Enumerations

- enum **Kind** {
  Kind::XYAscending, **Kind::XYDescending**, **Kind::YXAscending**, **Kind::YXDescending**,
  Kind::QualityAscending, **Kind::QualityDescending**, **Kind::AngleAscending**, **Kind::AngleDescending**,
  Kind::PolarCOMAscending, **Kind::PolarCOMDescending**, **Kind::PolarCOIAscending**, **Kind↩**
  ::PolarCOIDescending,
  Kind::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. 111) minutia.*
- std::vector< **Feature::MinutiaPoint** > **stableSort** (std::vector< **Feature::MinutiaPoint** > &minutia, const **Kind** &sortOrder)

  Sort *(p. 111) minutia, maintaining existing order if elements are otherwise deemed equal.*

F.5.1 Detailed Description

Utilities for sorting MinutiaPointSets.

F.5.2 Enumeration Type Documentation

F.5.2.1 Kind

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

Sort (p. 111) order of MinutiaPointSets.

Enumerator

XYAscending	Lowest to highest X value, followed by Y value.
XYDescending	Highest to lowest X value, followed by Y value.
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.
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).
PolarCOIDescending	Highest to lowest distance from center of img, followed by angle (theta).
Unknown	**Sort** (p. 111) order cannot be determined.

F.5.3 Function Documentation

F.5.3.1 sort()

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

Sort (p. 111) minutia.

Parameters

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

Exceptions

Error::NotImplemented *(p. 452)*	sortOrder is not implemented.
Error::StrategyError *(p. 560)*	Center of mass is specified, but no minutia.

F.5.3.2 stableSort()

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

Sort (p. 111) 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. 452)*	sortOrder is not implemented.
Error::StrategyError *(p. 560)*	Center of mass is specified, but no minutia.

F.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 **PatternClassification** {
  PlainArch = 0, **TentedArch**, **RadialLoop**, **UlnarLoop**,
  PlainWhorl, **CentralPocketLoop**, **DoubleLoop**, **AccidentalWhorl**,
  Whorl, **RightSlantLoop**, **LeftSlantLoop**, **Scar**,
  Amputation, **Unknown** }
- enum **Position** {
  Unknown = 0, **RightThumb** = 1, **RightIndex** = 2, **RightMiddle** = 3,
  RightRing = 4, **RightLittle** = 5, **LeftThumb** = 6, **LeftIndex** = 7,
  LeftMiddle = 8, **LeftRing** = 9, **LeftLittle** = 10, **PlainRightThumb** = 11,
  PlainLeftThumb = 12, **PlainRightFourFingers** = 13, **PlainLeftFourFingers** = 14, **LeftRightThumbs** = 15,
  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, **PlainRight↩ FourTips** = 51,
  PlainLeftFourTips = 52, **PlainRightFiveTips** = 53, **PlainLeftFiveTips** = 54 }

  Finger position codes.
- enum **Impression** {
  LiveScanPlain = 0, **LiveScanRolled** = 1, **NonLiveScanPlain** = 2, **NonLiveScanRolled** = 3,
  LatentImpression = 4, **LatentTracing** = 5, **LatentPhoto** = 6, **LatentLift** = 7,
  LiveScanVerticalSwipe = 8, **LiveScanPalm** = 10, **NonLiveScanPalm** = 11, **LatentPalmImpression** = 12,
  LatentPalmTracing = 13, **LatentPalmPhoto** = 14, **LatentPalmLift** = 15, **LiveScanOpticalContact↩ Plain** = 20,
  LiveScanOpticalContactRolled = 21, **LiveScanNonOpticalContactPlain** = 22, **LiveScanNonOptical↩ ContactRolled** = 23, **LiveScanOpticalContactlessPlain** = 24,
  LiveScanOpticalContactlessRolled = 25, **LiveScanNonOpticalContactlessPlain** = 26, **LiveScan↩ NonOpticalContactlessRolled** = 27, **Other** = 28,
  Unknown = 29 }
- enum **FingerImageCode** {
  EJI = 0, **RolledTip**, **FullFingerRolled**, **FullFingerPlainLeft**,
  FullFingerPlainCenter, **FullFingerPlainRight**, **ProximalSegment**, **DistalSegment**,
  MedialSegment, **NA** }

Functions

- std::ostream & **operator**<< (std::ostream &stream, const **AN2KViewCapture::FingerSegment↩ Position** &fsp)

  Output stream overload for FingerSegmentPosition.

F.6.1 Detailed Description

Biometric information relating to finger images and derived information.

The **Finger** (p. 113) 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.

F.6.2 Enumeration Type Documentation

F.6.2.1 FingerImageCode

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

Joint and tip codes.

F.6.2.2 Impression

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

Finger (p. 113), palm, and latent impression types.

F.6.2.3 PatternClassification

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

Pattern classification codes.

F.6.2.4 Position

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

Finger (p. 113) position codes.

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

F.7 BiometricEvaluation::Framework Namespace Reference

Information about the framework.

Classes

- class **API**

  A convenient way to execute biometric technology evaluation ***API*** (p. *220*) *methods safely.*
- class **Status**

Enumerations

- enum **APICurrentState** {
APICurrentState::NeverCalled, **APICurrentState::WatchdogExpired**, **APICurrentState::Signal↩ Caught**, **APICurrentState::ExceptionCaught**,
APICurrentState::Running, **APICurrentState::Completed** }

Functions

- unsigned int **getMajorVersion** ()

  Framework *(p. 115) major version.*
- unsigned int **getMinorVersion** ()

  Framework *(p. 115) 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. 159) 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. 559).*
- std::ostream & **operator<<** (std::ostream &s, const **Status** &status)

  Output stream operator overload.

F.7.1 Detailed Description

Information about the framework.

F.7.2 Enumeration Type Documentation

F.7.2.1 APICurrentState

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

Reasons operations could not complete.

Enumerator

NeverCalled	Operation was never executed.
WatchdogExpired	Watchdog timer expired.
SignalCaught	Signal handler was invoked.
ExceptionCaught	An exception was caught.
Running	Operation is running.
Completed	Operation has returned.

F.7.3 Function Documentation

F.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”

F.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.

F.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.

F.7.3.4 getCompileTime()

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

Time (p. 159) when this framework was compiled.

Returns

Time (p. 159) when this framework was compiled, in the form ”HH:MM:SS”

F.7.3.5 getMajorVersion()

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

Framework (p. 115) major version.

Returns

The major version number of the BiometricFramework

F.7.3.6 getMinorVersion()

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

Framework (p. 115) minor version.

Returns

The minor version of the **BiometricEvaluation** (p. 105) framework.

F.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. 559) object to output.

Returns

s appended with string representation of status.

F.7.3.8 to_string()

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

Obtain a textual representation of a **Status** (p. 559).

Parameters

status	**Status** (p. 559) object to convert.

Returns

Textual representation of status.

F.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. 533)), 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 **CompressionAlgorithm** {
  None = 0, **Facsimile** = 1, **WSQ20** = 2, **JPEGB** = 3,
  JPEGL = 4, **JP2** = 5, **JP2L** = 6, **PNG** = 7,
  NetPBM = 8, **BMP** = 9, **TIFF** = 10 }
- enum **PixelFormat** { **PixelFormat::MonoWhite** = 0, **PixelFormat::MonoBlack** = 1, **PixelFormat↩ ::Gray8** = 2, **PixelFormat::RGB24** = 3 }

Functions

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

  *Convert **Coordinate** (p. 283) 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. 542) 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. 526) 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. 533) 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

F.8.1 Detailed Description

Basic information relating to images.

Classes and methods for manipulating images.

The **Image** (p. 351) package gathers all image related matters, including classes to represent an image, coordinates, and functions for conversion between biometric representations.

F.8.2 Enumeration Type Documentation

F.8.2.1 CompressionAlgorithm

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

Image (p. 351) compression algorithms.

F.8.2.2 PixelFormat

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

Image (p. 351) 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

F.8.3 Function Documentation

F.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.

F.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.

F.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. 493) byte representation of an image.

Parameters

`in`	*bitDepth*	The number of bits that represents a single component in `rawData` (only 8 and 16 are supported).
`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. 470)*	Invalid `bitDepth` parameter.
BiometricEvaluation::Error::StrategyError *(p. 560)*	`rawData` does not appear to be sized large enough for the `bitsPerComponent` and `components` provided.

F.8.3.4 to_string() [1/5]

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

Convert **Coordinate** (p. 283) to std::string.

Parameters

c	**Coordinate** (p. 283) to convert to std::string.

Returns

std::string representation of c.

F.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.

F.8.3.6 to_string() [3/5]

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

Convert **Size** (p. 542) to std::string.

Parameters

s	**Size** (p. 542) to convert to std::string.

Returns

std::string representation of s.

F.8.3.7 to_string() [4/5]

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

Convert **Resolution** (p. 526) to std::string.

Parameters

r	**Resolution** (p. 526) to convert to std::string.

Returns

std::string representation of r.

F.8.3.8 to_string() [5/5]

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

Convert **ROI** (p. 533) to std::string.

Parameters

r	**ROI** (p. 533) to convert to std::string.

Returns

std::string representation of r.

F.8.4 Variable Documentation

F.8.4.1 CentimetersPerInch

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

Number of centimeters in one inch

F.8.4.2 MillimetersPerInch

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

Number of millimeters in one inch

F.9 BiometricEvaluation::IO Namespace Reference

Input/Output functionality.

Namespaces

- **Utility**

Classes

- class **ArchiveRecordStore**

  *This class implements the **IO::RecordStore** (p. 500) interface by storing data items in single file, with an associated manifest file.*
- class **CompressedRecordStore**

  *Sibling-implemented **RecordStore** (p. 500) with Compression.*
- class **Compressor**
- class **DBRecordStore**

  *A class that implements **IO::RecordStore** (p. 500) using a Berkeley DB database as the underlying record storage system.*
- class **FileLogCabinet**
- class **FileLogsheet**

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

  ***Compressor** (p. 269) for gzip compression from zlib.*
- class **ListRecordStore**

  ***RecordStore** (p. 500) that reads a list of keys from a text file, and retrieves the data from another **RecordStore** (p. 500).*
- class **Logsheet**

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

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

  *A **Properties** (p. 482) object persisted in an file on disk.*
- class **RecordStore**

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

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

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

  *A **RecordStore** (p. 500) 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 **Mode** { **Mode::ReadWrite** = 0, **Mode::ReadOnly** = 1 }

F.9.1 Detailed Description

Input/Output functionality.

The **IO** (p. 124) package contains classes and functions used to abstract input and output operations and provide for robust error handling on behalf of the application.

F.9.2 Enumeration Type Documentation

F.9.2.1 Mode

`enum 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.

F.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 &parent↩ Dir="/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.

F.10.1 Detailed Description

A class containing utility functions used for **IO** (p. 124) operations. These functions are class methods.

F.10.2 Function Documentation

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

Exceptions

Error::ObjectDoesNotExist (p. *453*)	The source named directory does not exist.
Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system, or the directoy name or prefix is malformed.

F.10.2.2 countLines() `[1/2]`

```
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. *312*)	Could not open path.

F.10.2.3 countLines() `[2/2]`

```
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.

F.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. *312*)	Could not create or close temporary file.
Error::MemoryError (p. *432*)	**Error** (p. 106) 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.

F.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. 312)*	Could not create or close temporary file.
Error::MemoryError *(p. 432)*	**Error** (p. 106) allocating memory for file name.

Returns

Open file stream to path.

Note

Caller must fclose(3) the returned stream.

F.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. 560)*	An error occurred when using the underlying storage system, or pathname is malformed.

F.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. 453)*	The named directory does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system, or pathname is malformed.

F.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. 129)

F.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. 129)

F.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.

F.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. 453)*	path does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

F.10.2.12 readPipe() `[1/2]`

```
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. 107) may contain more information.

F.10.2.13 readPipe() `[2/2]`

```
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.
pipeFD	The file descriptor of the pipe.

Exceptions

ObjectDoesNotExist	The reading end of the pipe has been closed.

Exceptions

FileError	The data could not be written in the entirety; **Error::errorStr()** (p. 107) may contain more information.

F.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. 453)*	The named directory does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system, or the directoy name or prefix is malformed.

F.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. 453)*	The named directory does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system, or the path name is malformed.

F.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. *453*)	The named object does not exist.
Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system, the name or prefix is malformed, or the maximum number of attempts was reached.

F.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. *453*)	The named directory does not exist.
Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system, or pathname is malformed.

F.10.2.18 writeFile() [1/2]

```
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.
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.

F.10.2.19 writeFile() [2/2]

```
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.
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.

F.10.2.20 writePipe() [1/2]

```
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. 107) may contain more information.

F.10.2.21 writePipe() [2/2]

```
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. 107) may contain more information.

F.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 **CaptureDeviceTechnology** { **Unknown** = 0, **CMOSCCD** = 1 }

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

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

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

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

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

  Range from camera lens center to subject iris.

F.11.1 Detailed Description

Biometric information relating to iris images and derived information.

The **Iris** (p. 136) 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.

F.12 BiometricEvaluation::Memory Namespace Reference

Support for memory-related operations.

Namespaces

- **AutoArrayUtility**

Classes

- class **AutoArray**

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

  RandomAccessIterator for any ***AutoArray*** *(p. 232).*
- 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. 115) 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. 115) 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. 115) 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)

F.12.1 Detailed Description

Support for memory-related operations.

The **Memory** (p. 137) package contains templates and classes that are used to manage memory, auto-sizing arrays, for example.

F.12.2 Function Documentation

F.12.2.1 isLittleEndian()

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

Determine endianess of current platform.

Returns

true if current platform is little endian. false otherwise.

F.12.2.2 make_unique() [1/3]

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

Framework (p. 115) 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.

F.12.2.3 make_unique() [2/3]

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

Framework (p. 115) 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.

F.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. 115) 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.

F.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.

F.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.

F.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.

F.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.

F.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.

F.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.

F.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. 232) 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. 232) 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.

F.13.1 Detailed Description

Convenience functions for AutoArrays.

F.13.2 Function Documentation

F.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. 232) of uint8_t or char to a char∗.

Parameters

rahc	**AutoArray** (p. 232) to cast.

Returns

rahc casted as a char∗.

F.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. 232) to a string.

Parameters

aa	**AutoArray** (p. 232) to stringify.
count	Last byte of aa to include in the returned string.

Returns

String representation of aa.

Exceptions

Error::MemoryError *(p. 432)*	count > aa.size()

F.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 std::string & str )  [inline]
```

Copy a string into an AutoAray of uint8_t or char.

Parameters

aa	**AutoArray** (p. 232) whose contents will be replaced with str.
str	String to assign to **AutoArray** (p. 232).

F.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 char * str,
              ... )  [inline]
```

Copy a string into an AutoAray of uint8_t or char.

Parameters

aa	**AutoArray** (p. 232) whose contents will be replaced with str.
str	printf-style format string.
...	Variable list of arguments for printf formatting.

F.14 BiometricEvaluation::MPI Namespace Reference

Common declarations and functions for the MPI-based functionality.

Classes

- class **CSVDistributor**
- 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. 143) task that distributes work to other tasks.*
- class **Exception**
- class **Receiver**

  *A class to represent an **MPI** (p. 143) task that receives WorkPackages containers from the **Distributor** (p. 303).*
- class **RecordProcessor**

  *An implementation of a work package processor that will extract record store keys, and optionally, values, from a **WorkPackage** (p. 600).*
- class **RecordStoreDistributor**

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

  *A class to represent a set of resources needed by an **MPI** (p. 143) program using a RecordStore for input.*
- class **Resources**
- class **Runtime**

  ***Runtime** (p. 533) support for the startup/shutdown of **MPI** (p. 143) 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. 303).*
- 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 **TaskCommand** : int32_t {
TaskCommand::Continue = 0, **TaskCommand::Ignore** = 1, **TaskCommand::Exit** = 2, **Task↩ Command::QuickExit** = 3,
TaskCommand::TermExit = 4 }
- enum **TaskStatus** : int32_t { **TaskStatus::OK** = 0, **TaskStatus::Failed** = 1, **TaskStatus::Exit** = 2, **TaskStatus::RequestJobTermination** = 3 }
- enum **MessageTag** : int32_t { **MessageTag::Control** = 0, **MessageTag::Data** = 1, **MessageTag::↩ 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. 143) framework.*

Variables

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

F.14.1 Detailed Description

Common declarations and functions for the MPI-based functionality.

F.14.2 Typedef Documentation

F.14.2.1 msgtag_t

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

Storage type for MessageTag.

F.14.2.2 taskcmd_t

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

Storage type for TaskCommand.

F.14.2.3 taskstat_t

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

Storage type for TaskStatus.

F.14.3 Enumeration Type Documentation

F.14.3.1 MessageTag

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

The types of messages sent between **MPI** (p. 143) 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.

F.14.3.2 TaskCommand

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

The command given to an **MPI** (p. 143) task.

Enumerator

Continue	Normal operation.
Ignore	Ignore the message.
Exit	Transition to the normal shutdown state.
QuickExit	Transition to the quick shutdown state.
TermExit	Transition to the immeditate shutdown state.

F.14.3.3 TaskStatus

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

The status of an **MPI** (p. 143) distributor or receiver task.

Enumerator

OK	Normal operation.
Failed	Failed to complete an operation.
Exit	Transitioned to the shutdown state.
RequestJobTermination	Requesting that **Distributor** (p. 303) stops the job.

F.14.4 Function Documentation

F.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. 143) rank, and process ID, formatted in a manner that can be used to uniquely name files.

Returns

The unique ID for the process.

F.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. 143) job, errors are managed, and therefore, log messages may stop if the Logsheet has failed.

Parameters

in	*logsheet*	The open Logsheet to write into.

F.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. 143) 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.

F.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. 143) 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. 470)*	Invalid URL.
Error::Exception *(p. 307)*	Failed to create the Logsheet object. The exception string will contain more information.

F.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.

F.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. 147) view and derived information.*

Enumerations

- enum **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, **RightWristBracelet** = 85,
LeftWristBracelet = 86 }

Palm position codes.

F.15.1 Detailed Description

Biometric information relating to palm images and derived information.

The **Palm** (p. 147) package gathers all palm related matters,

F.15.2 Enumeration Type Documentation

F.15.2.1 Position

`enum BiometricEvaluation::Palm::Position [strong]`

Palm (p. 147) position codes.

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

F.16 BiometricEvaluation::Plantar Namespace Reference

Biometric information relating to plantar images and derived information.

Enumerations

- enum **Position** {
  UnknownSole = 60, **RightSole** = 61, **LeftSole** = 62, **UnknownToe** = 63,
  RightBigToe = 64 }

  Plantar position codes.

F.16.1 Detailed Description

Biometric information relating to plantar images and derived information.

F.16.2 Enumeration Type Documentation

F.16.2.1 Position

`enum BiometricEvaluation::Plantar::Position [strong]`

Plantar (p. 148) position codes.

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

F.17 BiometricEvaluation::Process Namespace Reference

Process (p. 148) information and controls.

Classes

- class **CommandCenter**
- class **CommandParser**
- class **ForkManager**

  Manager *(p. 426) implementation that starts Workers by calling fork(2).*
- class **ForkWorkerController**

  Wrapper of a ***Worker*** *(p. 588) returned from a* ***Process::ForkManager*** *(p. 331).*
- 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. 432).*
- class **POSIXThreadManager**

  Manager *(p. 426) implementation that starts Workers in POSIX threads.*
- class **POSIXThreadWorkerController**

  Decorated ***Worker*** *(p. 588) returned from a* ***Process::POSIXThreadManager*** *(p. 476).*
- class **Semaphore**

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

  The ***Statistics*** *(p. 554) 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. 588) returned from a* ***Process::Manager*** *(p. 426).*

Typedefs

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

F.17.1 Detailed Description

Process (p. 148) information and controls.

The **Process** (p. 148) package gathers all process related matters, including a class to obtain resource usage statistics.

F.17.2 Typedef Documentation

F.17.2.1 ParameterList

```
using BiometricEvaluation::Process::ParameterList = typedef std::map<std::string, std::shared↩
_ptr<void> >
```

Convenience alias for parameter lists to child routines

F.18 BiometricEvaluation::System Namespace Reference

Operating system, hardware, etc.

Functions

- uint32_t **getCPUCount** ()

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

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

  Obtain the system load average for the last minute.

F.18.1 Detailed Description

Operating system, hardware, etc.

The **System** (p. 150) package gathers all system related matters, such as the operating system name, number of CPUs, etc.

F.18.2 Function Documentation

F.18.2.1 getCPUCount()

`uint32_t BiometricEvaluation::System::getCPUCount ( )`

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

Returns

The number of processing units.

Exceptions

Error::NotImplemented *(p. 452)*	Not implemented for this operating system, or the underlying OS feature is not installed.

F.18.2.2 getLoadAverage()

`double BiometricEvaluation::System::getLoadAverage ( )`

Obtain the system load average for the last minute.

Returns

The system load average.

Exceptions

Error::NotImplemented *(p. 452)*	Not implemented for this operating system, or the underlying OS feature is not installed.

F.18.2.3 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. 452)*	Not implemented for this operating system, or the underlying OS feature is not installed.

F.19 BiometricEvaluation::Text Namespace Reference

Text (p. 151) 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.

F.19.1 Detailed Description

Text (p. 151) processing for string objects.

The **Text** (p. 151) package contains a set of functions for the processing of strings: removing leading and trailing whitespace, computing a digest, and other utility functions.

F.19.2 Function Documentation

F.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.

F.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.

F.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.

F.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. *432*)	Could not allocate memory to store digest.
Error::NotImplemented (p. *452*)	The value of digest is not a supported digest.
Error::StrategyError (p. *560*)	An error occurred while obtaining the digest.

Returns

An ASCII representation of the hex digits composing the digest.

F.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. 432)*	Could not allocate memory to store digest.
Error::NotImplemented *(p. 452)*	The value of digest is not a supported digest.
Error::StrategyError *(p. 560)*	An error occurred while obtaining the digest.

Returns

An ASCII representation of the hex digits composing the digest.

F.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.

F.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.

F.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 trimChar should be removed.
trimChar	Character to remove from the beginning of s.

Returns

Copy of s without leading trimChar.

F.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.

F.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 trimChar should be removed.
trimChar	Character to remove from the end of s.

Returns

Copy of s without trailing trimChar.

F.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.
locale	Locale to be considered when determining whitespace characters.

Returns

Copy of s without trailing whitespace.

F.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 '\'.
`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.

F.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 loercase.
locale	Locale to use when lowercasing str.

Returns

Lowercase copy of str.

F.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.

F.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 trimChar should be removed.
trimChar	Character to remove from the beginning and ending of s.

Returns

Copy of s without leading or trailing trimChar.

F.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.

F.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. 585) 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. 571).*
- 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

F.20.1 Detailed Description

Support for time and timers.

The **Time** (p. 159) package gathers all timing relating matters, such as Timers, **Watchdog** (p. 585) timers, etc. **Time** (p. 159) values are in microsecond units.

F.20.2 Function Documentation

F.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.

F.20.2.2 getCurrentDate()

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

Returns

The current ISO 8601 date as a string.

F.20.2.3 getCurrentDateAndTime()

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

Returns

The standard locale current date and time as a string.

F.20.2.4 getCurrentTime()

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

Returns

The current ISO 8601 time as a string.

F.20.2.5 operator<<()

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

Output stream operator overload for **Timer** (p. 571).

Parameters

s	Stream to append.
timer	**Timer** (p. 571) whose elapsed time in microseconds should be appended to s.

Returns

s with value of elapsedStr() appended.

Exceptions

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

F.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`)

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

F.21.1 Detailed Description

Basic information relating to video and streams.

Common representation of a video stream. **Stream** (p. 561) objects can only be obtained from **Container** (p. 280) objects.

The **Video** (p. 162) package gathers all video related matters, including classes to represent a video stream and video containers.

F.21.2 Enumeration Type Documentation

F.21.2.1 CodingFormat

`enum BiometricEvaluation::Video::CodingFormat [strong]`

Video (p. 162) coding formats.

F.21.2.2 ContainerFormat

`enum BiometricEvaluation::Video::ContainerFormat [strong]`

Container (p. 280) formats

F.22 BiometricEvaluation::View Namespace Reference

View (p. 581) 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.

F.22.1 Detailed Description

View (p. 581) information.

The **View** (p. 581) 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.

F.22.2 Function Documentation

F.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 AN2KQualityMetric information.
in	*qm*	AN2KQualityMetric information to append to stream.

Returns

stream with a qm textual representation appended.

F.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.

Parameters

`in`	*ppc*	PrintPositionCoordinate information to append to stream.

Returns

Stream with a ppc textual representation appended.

Appendix G

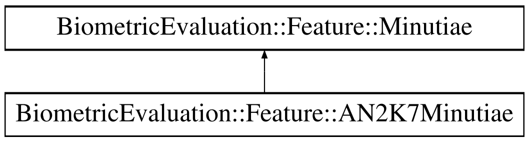
Class Documentation

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

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

`#include <be_feature_an2k7minutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::AN2K7Minutiae:


BiometricEvaluation::Feature::Minutiae
BiometricEvaluation::Feature::AN2K7Minutiae

Classes

- struct **FingerprintReadingSystem**

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

  Pattern classification codes.

Public Types

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

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

Public Member Functions

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

  *Construct an AN2K7 **Minutiae** (p. 438) object from file data.*
- **AN2K7Minutiae** (**Memory::uint8Array** &buf, int recordNumber)

  *Construct an AN2K7 **Minutiae** (p. 438) object from data contained in a memory buffer.*
- PatternClassificationSet **getPatternClassificationSet** () const

Obtain the set fingerprint pattern classifications.

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

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

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

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

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

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

Static Public Member Functions

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

  *Convert string read from AN2K record into a **PatternClassification** (p. 470).*
- static **Finger::PatternClassification convertPatternClassification** (const **PatternClassification::**↩ **Entry** &entry)

  *Convert a standard **PatternClassification::Entry** (p. 306) 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.

G.1.1 Detailed Description

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

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

G.1.2 Member Enumeration Documentation

G.1.2.1 EncodingMethod

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

Methods for encoding minutiae data in an AN2K record.

Enumerator

Automatic	No possible human interaction
AutomaticUnedited	Editing possible, but not performed
AutomaticEdited	Editing possible and was performed

G.1.3 Constructor & Destructor Documentation

G.1.3.1 AN2K7Minutiae() `[1/2]`

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

Construct an AN2K7 **Minutiae** (p. 438) 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. 312)*	An error occurred when opening or reading from the file.
Error::DataError *(p. 293)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.

G.1.3.2 AN2K7Minutiae() `[2/2]`

```
BiometricEvaluation::Feature::AN2K7Minutiae::AN2K7Minutiae (
              Memory::uint8Array & buf,
             int recordNumber )
```

Construct an AN2K7 **Minutiae** (p. 438) 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. 293)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.

G.1.4 Member Function Documentation

G.1.4.1 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. 283) representation of str.

Exceptions

Error::DataError *(p. 293)*	Invalid format of str.

G.1.4.2 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. 293)*	Invalid value for mem.

G.1.4.3 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. 470).

Parameters

in	*fpc*	Value for pattern classification read from AN2K record.

Exceptions

Error::DataError *(p. 293)*	Invalid value for fpc.

G.1.4.4 convertPatternClassification() [2/2]

```
static  Finger::PatternClassification BiometricEvaluation::Feature::AN2K7Minutiae::convert↩
PatternClassification (
            const  PatternClassification::Entry & entry )  [static]
```

Convert a standard **PatternClassification::Entry** (p. 306) to a PatternClassification::Kind.

Parameters

in	*entry*	A standard pattern classification entry

Exceptions

Error::DataError *(p. 293)*	Non-standard pattern classification entry.

G.1.4.5 getOriginatingFingerprintReadingSystem()

```
 FingerprintReadingSystem BiometricEvaluation::Feature::AN2K7Minutiae::getOriginatingFingerprint↩
ReadingSystem ( ) const
```

Obtain the originating fingerprint reading system.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	The optional OFR field has been excluded.

G.1.4.6 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.

G.2 BiometricEvaluation::Finger::AN2KMinutiaeDataRecord Class Reference

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

```
#include <be_finger_an2kminutiae_data_record.h>
```

Public Member Functions

- **AN2KMinutiaeDataRecord** (const std::string &filename, int recordNumber)

  Construct an ***AN2KMinutiaeDataRecord*** (p. *169*) *object from data contained in a file on disk.*
- **AN2KMinutiaeDataRecord** (**Memory::uint8Array** &buf, int recordNumber)

  Construct an ***AN2KMinutiaeDataRecord*** (p. *169*) *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.

G.2.1 Detailed Description

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

Type-9 Records may contain only "standard" minutiae data (fields 9.005 - 9.012) or any combination of "standard" minutiae data, 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.

G.2.2 Constructor & Destructor Documentation

G.2.2.1 AN2KMinutiaeDataRecord() [1/2]

```
BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::AN2KMinutiaeDataRecord (
            const std::string & filename,
            int recordNumber )
```

Construct an **AN2KMinutiaeDataRecord** (p. 169) 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. 312)*	An error occurred when opening or reading from the file.
Error::DataError *(p. 293)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.

G.2.2.2 AN2KMinutiaeDataRecord() [2/2]

```
BiometricEvaluation::Finger::AN2KMinutiaeDataRecord::AN2KMinutiaeDataRecord (
              Memory::uint8Array & buf,
             int recordNumber )
```

Construct an **AN2KMinutiaeDataRecord** (p. 169) 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. 293)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.

G.2.3 Member Function Documentation

G.2.3.1 getAN2K11EFS()

```
std::shared_ptr< Feature::AN2K11EFS::ExtendedFeatureSet> BiometricEvaluation::Finger::AN2K↩
MinutiaeDataRecord::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.

G.2.3.2 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.

G.2.3.3 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.

G.2.3.4 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. 452)*	Cannot return a map of fields for vendor, likely because there exists a better, native implementation of accessing minutiae data in **AN2KMinutiaeDataRecord** (p. 169).

G.3 BiometricEvaluation::View::AN2KViewVariableResolution::AN2←KQualityMetric 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**

G.3.1 Detailed Description

A structure to represent an AN2K quality metric.

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

G.4 BiometricEvaluation::DataInterchange::AN2KRecord Class Reference

A class to represent an entire ANSI/NIST record.

`#include <be_data_interchange_an2k.h>`

Classes

- struct **CharacterSet**
- struct **DomainName**

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

Public Types

- using **DomainName** = struct **DomainName**
- using **CharacterSet** = struct **CharacterSet**

Public Member Functions

- **AN2KRecord** (const std::string filename)

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

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

  Obtain the count of latent (Type-13) finger views.
- std::vector< **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.
- 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::Record↩ Type** recordType)

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

G.4.1 Detailed Description

A class to represent an entire ANSI/NIST record.

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

G.4.2 Member Typedef Documentation

G.4.2.1 CharacterSet

`using BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet = struct CharacterSet`

Convenience alias for struct **CharacterSet** (p. 253)

G.4.2.2 DomainName

`using BiometricEvaluation::DataInterchange::AN2KRecord::DomainName = struct DomainName`

Convenience alias for struct **DomainName** (p. 305)

G.4.3 Constructor & Destructor Documentation

G.4.3.1 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

Exceptions

Error::FileError *(p. 312)*	An error occurred when opening or reading the file.
Error::DataError *(p. 293)*	An error occurred when processing the AN2K record.

G.4.3.2 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. 293)*	An error occurred when processing the AN2K record.

G.4.4 Member Function Documentation

G.4.4.1 getDate()

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

Returns

The date field in the Type-1 record.

G.4.4.2 getDestinationAgency()

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

Returns

The destination agency ID.

G.4.4.3 getDirectoryOfCharacterSets()

`std::vector< CharacterSet> BiometricEvaluation::DataInterchange::AN2KRecord::getDirectoryOf←
CharacterSets ( ) const`

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

Returns

Vector of **CharacterSet** (p. 253) structs representing other character sets that may appear in the transaction.

G.4.4.4 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. 305) struct with identifier and version information (if defined).

G.4.4.5 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.

G.4.4.6 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. 173).

Returns

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

G.4.4.7 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.

G.4.4.8 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. 173).

Returns

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

G.4.4.9 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.

G.4.4.10 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.

G.4.4.11 getNativeScanningResolution()

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

Returns

The native scanning resolution.

G.4.4.12 getNominalTransmittingResolution()

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

Returns

The nominal transmitting resolution.

G.4.4.13 getOriginatingAgency()

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

Returns

The originating agency ID.

G.4.4.14 getPriority()

`uint8_t BiometricEvaluation::DataInterchange::AN2KRecord::getPriority ( ) const`

Obtain the urgency with which a response is required.

Returns

Priority (1:High - 9:Low)

G.4.4.15 getTransactionControlNumber()

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

Returns

The transcantion control number.

G.4.4.16 getVersionNumber()

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

Returns

The record version field in the Type-1 record.

G.4.4.17 recordLocations() [1/2]

```
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. 293)*	An error occurred when processing the AN2K record.

G.4.4.18 recordLocations() `[2/2]`

```
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

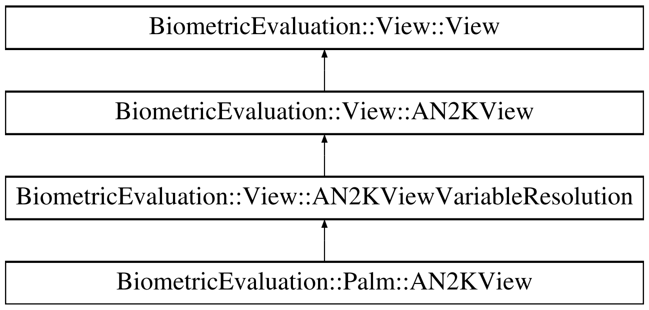
Set of integer positions within the ANSI_NIST struct where a recordType Record is located.

G.5 BiometricEvaluation::Palm::AN2KView Class Reference

A class to represent a single **Palm** (p. 147) view and derived information.

`#include <be_palm_an2kview.h>`

Inheritance diagram for BiometricEvaluation::Palm::AN2KView:


BiometricEvaluation::View::View
BiometricEvaluation::View::AN2KView
BiometricEvaluation::View::AN2KViewVariableResolution
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.

Additional Inherited Members

G.5.1 Detailed Description

A class to represent a single **Palm** (p. 147) view and derived information.

A **Palm::AN2KView** (p. 179) 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.

G.5.2 Constructor & Destructor Documentation

G.5.2.1 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.

G.5.2.2 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.

G.5.3 Member Function Documentation

G.5.3.1 getPalmQualityMetric()

```
QualityMetricSet BiometricEvaluation::Palm::AN2KView::getPalmQualityMetric ( ) const
```

Obtain the palm quality metric.

Returns

QualityMetricSet containing the set of metrics the palm image.

G.5.3.2 getPosition()

`Palm::Position BiometricEvaluation::Palm::AN2KView::getPosition ( ) const`

Obtain the palm position.

Returns

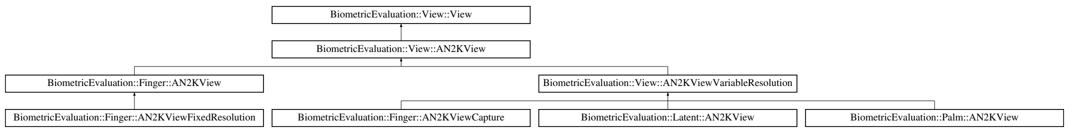
The palm position.

G.6 BiometricEvaluation::View::AN2KView Class Reference

A class to represent single biometric view and derived information.

`#include <be_view_an2kview.h>`

Inheritance diagram for BiometricEvaluation::View::AN2KView:


BiometricEvaluation::View::View
BiometricEvaluation::View::AN2KView
BiometricEvaluation::Finger::AN2KView
BiometricEvaluation::View::AN2KViewVariableResolution
BiometricEvaluation::Finger::AN2KViewFixedResolution
BiometricEvaluation::Finger::AN2KViewCapture
BiometricEvaluation::Latent::AN2KView
BiometricEvaluation::Palm::AN2KView

Public Types

- enum **RecordType** : uint16_t {
  Type_1 = 1, **Type_2** = 2, **Type_3** = 3, **Type_4** = 4,
  Type_5 = 5, **Type_6** = 6, **Type_7** = 7, **Type_8** = 8,
  Type_9 = 9, **Type_10** = 10, **Type_11** = 11, **Type_12** = 12,
  Type_13 = 13, **Type_14** = 14, **Type_15** = 15, **Type_16** = 16,
  Type_17 = 17, **Type_99** = 99 }
- enum **DeviceMonitoringMode** {
  DeviceMonitoringMode::Controlled, **DeviceMonitoringMode::Assisted**, **DeviceMonitoringMode**↩
  ::Observed, **DeviceMonitoringMode::Unattended**,
  DeviceMonitoringMode::Unknown, **DeviceMonitoringMode::NA** }

  The level of human monitoring for the image capture device.

Public Member Functions

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a file.
- **AN2KView** (**Memory::uint8Array** &buf, const **RecordType** typeID, const uint32_t recordNumber)

  Construct an AN2K view from a buffer.
- std::vector< **Finger::AN2KMinutiaeDataRecord** > **getMinutiaeDataRecordSet** () const

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

  Obtain the ANSI-NIST record type.

Static Public Member Functions

- static **DeviceMonitoringMode** **convertDeviceMonitoringMode** (const char ∗dmm)

  Convert a device monitoring mode indicator from an AN2K record.

- static **Image::CompressionAlgorithm** **convertCompressionAlgorithm** (const uint16_t recordType, const unsigned char ∗an2kValue)

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

Static Public Attributes

- static const double **MinimumScanResolutionPPMM**

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

  The defined bit-depth for fixed-resolution images.

Protected Member Functions

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

  Obtain the complete ANSI/NIST record set.
- RECORD ∗ **getAN2KRecord** () const

  Obtain a pointer to the single ANSI/NIST record.

G.6.1 Detailed Description

A class to represent single biometric view and derived information.

This abstraction represents the image and derived information taken from an ANSI/NIST record.

For these types of records, the image resolution and scan resolution are identical. For compressed images, applications can compare the image resolution and size taken from the Type-3/4/5/6 record to that returned by the **Image** (p. 118) object directly.

G.6.2 Member Enumeration Documentation

G.6.2.1 DeviceMonitoringMode

`enum BiometricEvaluation::View::AN2KView::DeviceMonitoringMode [strong]`

The level of human monitoring for the image capture device.

Enumerator

Controlled	Operator physically controls the subject to acquire biometric sample.
Assisted	Person available to provide assistance to the subject submitting the biometric.
Observed	Person present to observe the operation of the device but provides no assistance.
Unattended	No one present to observe or provide assistance.
Unknown	No information is known.
NA	Optional field – not specified

G.6.2.2 RecordType

```
enum  BiometricEvaluation::View::AN2KView::RecordType :  uint16_t  [strong]
```

The type of AN2K record.

G.6.3 Constructor & Destructor Documentation

G.6.3.1 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.

G.6.3.2 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.

G.6.4 Member Function Documentation

G.6.4.1 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. 293)*	Invalid compression algorithm for record type.

Exceptions

Error::ParameterError *(p. 470)*	Invalid record type.

G.6.4.2 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. 293)*	Invalid format of dmm.

G.6.4.3 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.

G.6.4.4 getMinutiaeDataRecordSet()

```
std::vector< Finger::AN2KMinutiaeDataRecord> BiometricEvaluation::View::AN2KView::getMinutiae↩
DataRecordSet ( ) const
```

Obtain the set of minutiae records.

Each **AN2KViewVariableResolution** (p. 198) 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.

G.6.4.5 getRecordType()

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

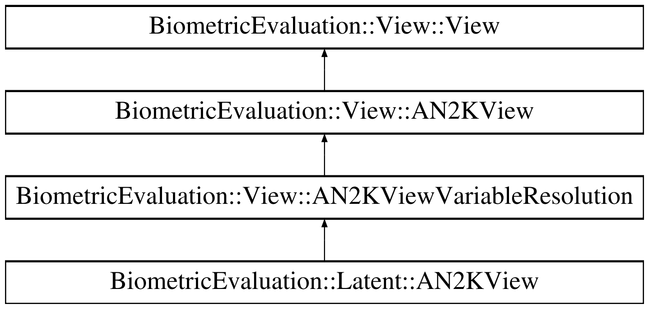
Obtain the ANSI-NIST record type.

Returns

The type of record used to construct this object.

G.7 BiometricEvaluation::Latent::AN2KView Class Reference

Inheritance diagram for BiometricEvaluation::Latent::AN2KView:


BiometricEvaluation::View::View
BiometricEvaluation::View::AN2KView
BiometricEvaluation::View::AN2KViewVariableResolution
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.

Additional Inherited Members

G.7.1 Constructor & Destructor Documentation

G.7.1.1 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.

G.7.1.2 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.

G.7.2 Member Function Documentation

G.7.2.1 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

G.7.2.2 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.

G.7.2.3 getPrintPositionCoordinates()

```
PrintPositionCoordinateSet BiometricEvaluation::Latent::AN2KView::getPrintPositionCoordinates
( ) const
```

Obtain print position coordinates.

Returns

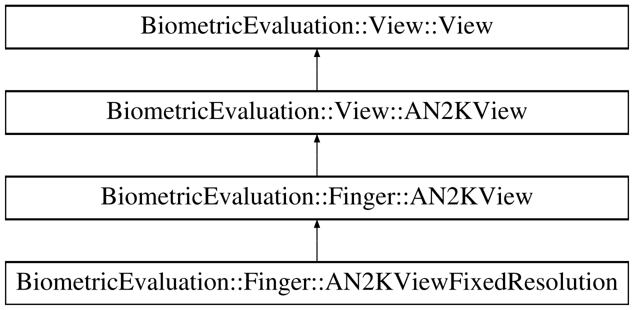
Set of all PrintPositionCoordinates

G.8 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:


BiometricEvaluation::View::View
BiometricEvaluation::View::AN2KView
BiometricEvaluation::Finger::AN2KView
BiometricEvaluation::Finger::AN2KViewFixedResolution

Public Member Functions

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

  Obtain the set of minutiae records.
- Finger::PositionSet **getPositions** () const

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

  Obtain the finger impression code.

Static Public Member Functions

- static **Finger::Position convertPosition** (int an2kFGP)

  Convert a compression algorithm indicator from an AN2K finger image record.
- static Finger::PositionSet **populateFGP** (FIELD ∗field)

  Read the finger positions from an AN2K record.
- static **Finger::Impression convertImpression** (const unsigned char ∗str)

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

  Convert an finger image code from a string.

Protected Member Functions

- **AN2KView** (const std::string filename, const **RecordType** typeID, const uint32_t recordNumber)

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

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

  *Add a minutiae data record to the **AN2KMinutiaeDataRecord** (*p. 169*) set.*
- void **setPositions** (Finger::PositionSet &ps)

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

  Mutator for the impression type.

Additional Inherited Members

G.8.1 Detailed Description

A class to represent single finger view and derived information.

A base **Finger::AN2KView** (p. 186) 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. 118) object directly.

G.8.2 Constructor & Destructor Documentation

G.8.2.1 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. 470)*	An invalid parameter was passed in.
Error::DataError *(p. 293)*	An error occurred when parsing the AN2K record.
Error::FileError *(p. 312)*	An error occurred when reading the file.

G.8.2.2 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. 470)*	An invalid parameter was passed in.
Error::DataError *(p. 293)*	An error occurred when parsing the AN2K record.

G.8.3 Member Function Documentation

G.8.3.1 addMinutiaeDataRecord()

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

Add a minutiae data record to the **AN2KMinutiaeDataRecord** (p. 169) set.

Parameters

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

G.8.3.2 convertFingerImageCode()

```
static  Finger::FingerImageCode BiometricEvaluation::Finger::AN2KView::convertFingerImageCode
(
             const char * str )  [static]
```

Convert an finger image code from a string.

Parameters

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

Returns

A FingerImageCode value.

Exceptions

Error::DataError *(p. 293)*	The string contains an invalid image code.

G.8.3.3 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. 293)*	The position code is invalid.

G.8.3.4 getImpressionType()

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

Obtain the finger impression code.

Returns

The finger impression code.

G.8.3.5 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.

G.8.3.6 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.

G.8.3.7 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. *293)*	The data contains an invalid value.

G.8.3.8 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.

G.8.3.9 setPositions()

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

Add a position set to the collection of position sets.

Parameters

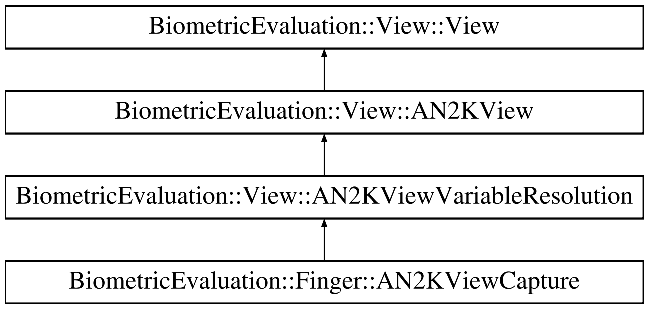
in	*ps*	The position set to be added.

G.9 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:


BiometricEvaluation::View::View
BiometricEvaluation::View::AN2KView
BiometricEvaluation::View::AN2KViewVariableResolution
BiometricEvaluation::Finger::AN2KViewCapture

Classes

- struct **FingerSegmentPosition**

  Locations of an individual finger segment in a slap.

Public Types

- enum **AmputatedBandaged** { **AmputatedBandaged::Amputated**, **AmputatedBandaged::Bandaged**, **AmputatedBandaged::NA** }

  Enumeration of the finger amuptated or bandaged code, a reason that a capture could not be made.
- using **FingerSegmentPosition** = struct **FingerSegmentPosition**
- using **FingerSegmentPositionSet** = std::vector< **FingerSegmentPosition** >

Public Member Functions

- **AN2KViewCapture** (const std::string &filename, const uint32_t recordNumber)

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

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

  Extract the NQM information from an AN2K FIELD.
- **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.

Additional Inherited Members

G.9.1 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).

G.9.2 Member Enumeration Documentation

G.9.2.1 AmputatedBandaged

```
enum BiometricEvaluation::Finger::AN2KViewCapture::AmputatedBandaged [strong]
```

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

Enumerator

Amputated	Amputation
Bandaged	Unable to print (e.g., bandaged)
NA	Optional field – not specified

G.9.3 Constructor & Destructor Documentation

G.9.3.1 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. 470)	
Error::DataError (p. 293)	
Error::FileError (p. 312)	An error occurred when opening or reading the file.

G.9.3.2 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.

G.9.4 Member Function Documentation

G.9.4.1 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. 293)*	Invalid format of field for NQM.

G.9.4.2 getAlternateFingerSegmentPositionSet()

```
FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getAlternateFingerSegment←
PositionSet ( ) const
```

Returns

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

G.9.4.3 getAmputatedBandaged()

```
AmputatedBandaged BiometricEvaluation::Finger::AN2KViewCapture::getAmputatedBandaged ( ) const
```

Returns

Optional amputated or bandaged code.

G.9.4.4 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

G.9.4.5 getFingerSegmentPositionSet()

```
FingerSegmentPositionSet BiometricEvaluation::Finger::AN2KViewCapture::getFingerSegmentPosition←
Set ( ) const
```

Returns

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

G.9.4.6 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.

G.9.4.7 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.

G.9.4.8 getPrintPositionCoordinates()

```
PrintPositionCoordinateSet BiometricEvaluation::Finger::AN2KViewCapture::getPrintPositionCoordinates
( ) const
```

Obtain print position coordinates.

Returns

Set of all PrintPositionCoordinates

G.9.4.9 getSegmentationQualityMetric()

```
QualityMetricSet BiometricEvaluation::Finger::AN2KViewCapture::getSegmentationQualityMetric (
) const
```

Obtain the segmentation quality metric for all segmented finger images.

Returns

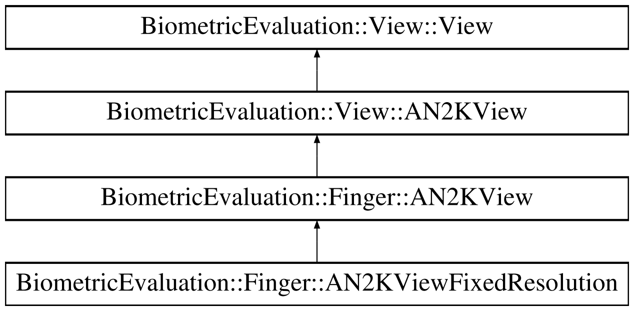
QualityMetricSet containing the segmentation quality metric for all segmented finger images.

G.10 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:


BiometricEvaluation::View::View
BiometricEvaluation::View::AN2KView
BiometricEvaluation::Finger::AN2KView
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.

Additional Inherited Members

G.10.1 Detailed Description

A class to represent single finger view and derived information.

A base **Finger::AN2KView** (p. 186) 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. 118) object directly.

G.10.2 Constructor & Destructor Documentation

G.10.2.1 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. 470)*	An invalid parameter was passed in.
Error::DataError *(p. 293)*	An error occurred when parsing the AN2K record.
Error::FileError *(p. 312)*	An error occurred when reading the file.

G.10.2.2 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.
in	*recordNumber*	Which finger record to read as there may be multiple finger views of the same type within a single AN2K record.

Exceptions

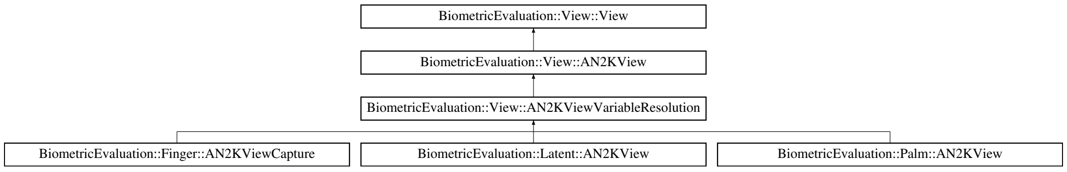
Error::ParameterError *(p. 470)*	An invalid parameter was passed in.
Error::DataError *(p. 293)*	An error occurred when parsing the AN2K record.

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

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

`#include <be_view_an2kview_varres.h>`

Inheritance diagram for BiometricEvaluation::View::AN2KViewVariableResolution:


BiometricEvaluation::View::View
BiometricEvaluation::View::AN2KView
BiometricEvaluation::View::AN2KViewVariableResolution
BiometricEvaluation::Finger::AN2KViewCapture
BiometricEvaluation::Latent::AN2KView
BiometricEvaluation::Palm::AN2KView

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** = struct **PrintPositionCoordinate**
- using **PrintPositionCoordinateSet** = std::vector< **PrintPositionCoordinate** >

Public Member Functions

- **Finger::Impression getImpressionType** () const
- std::string **getSourceAgency** () const
- std::string **getCaptureDate** () const
- std::string **getComment** () const

  Obtain the comment field.
- **Memory::uint8Array getUserDefinedField** (const uint16_t field) const

  Obtain a user-defined field.

Static Public Member Functions

- static QualityMetricSet **extractQuality** (FIELD ∗field, **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.

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

  Obtain the set of finger positions.
- 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.

Additional Inherited Members

G.11.1 Detailed Description

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

The view represents a variable resolution (Type-13/14/15) AN2K record.

G.11.2 Constructor & Destructor Documentation

G.11.2.1 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.

G.11.2.2 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.

G.11.3 Member Function Documentation

G.11.3.1 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. 293)*	The data contains an invalid value.

G.11.3.2 getCaptureDate()

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

Returns

The capture date.

G.11.3.3 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.

G.11.3.4 getImpressionType()

```
Finger::Impression BiometricEvaluation::View::AN2KViewVariableResolution::getImpressionType
( ) const
```

Returns

The finge/palmr impression code.

G.11.3.5 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.

G.11.3.6 getPositions()

```
Feature::FGPSet BiometricEvaluation::View::AN2KViewVariableResolution::getPositions ( ) const
[protected]
```

Obtain the set of finger positions.

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.

G.11.3.7 getPrintPositionCoordinates()

```
PrintPositionCoordinateSet BiometricEvaluation::View::AN2KViewVariableResolution::getPrint↩
PositionCoordinates ( ) const  [protected]
```

Obtain print position coordinates.

Returns

Set of all PrintPositionCoordinates

G.11.3.8 getQualityMetric()

```
QualityMetricSet BiometricEvaluation::View::AN2KViewVariableResolution::getQualityMetric ( )
const  [protected]
```

Obtain quality metrics for associated image record.

Returns

Quality metrics

G.11.3.9 getSourceAgency()

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

Returns

The source agency.

G.11.3.10 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. 453)	There is no user-defined field with the requested field number.
Error::ParameterError (p. 470)	Invalid value for field.
Error::StrategyError (p. 560)	Field could not be cached.

G.11.3.11 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. 453)	There is no user-defined field with the requested field number.
Error::ParameterError (p. 470)	Invalid value for fieldID.

G.12 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

G.12.1 Detailed Description

Sort (p. 111) by increasing angle (theta)

G.12.2 Member Function Documentation

G.12.2.1 operator()()

```
bool BiometricEvaluation::Feature::Sort::Angle::operator() (
             const  BiometricEvaluation::Feature::MinutiaPoint & lhs,
             const  BiometricEvaluation::Feature::MinutiaPoint & rhs ) const
```

MinutiaPoint (p. 440) angle ascending comparator.

G.13 BiometricEvaluation::DataInterchange::ANSI2004Record Class Reference

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

Public Member Functions

- **ANSI2004Record** (const BE::Memory::uint8Array &fmr, const BE::Memory::uint8Array &fir)

  ANSI2004Record *(p. 203) constructor using a pair of finger minutia and image records.*
- **ANSI2004Record** (const std::string &fmrPath, const std::string &firPath)

  ANSI2004Record *(p. 203) constructor using a pair of finger minutia and image records.*
- **ANSI2004Record** (const std::initializer_list< BE::Finger::ANSI2004View > &views)

  ANSI2004Record *(p. 203) 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< BE::Feature::INCITSMinutiae > **getMinutia** () const

  Obtain the INCITSMinutiae for all finger views.
- BE::Feature::INCITSMinutiae **getMinutia** (uint32_t viewNumber) const

  Obtain the INCITSMinutiae for a finger view.
- void **setMinutia** (const std::vector< BE::Feature::INCITSMinutiae > &minutia)

  Alter the minutia for every finger view.
- void **setMinutia** (uint32_t viewNumber, const BE::Feature::INCITSMinutiae &minutia)

  Alter the minutia for a single finger view.
- BE::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. 205).*
- uint64_t **getEDBLength** () const

  Obtain the size of EDB that will be written by ***getFMR()*** *(p. 205).*

G.13.1 Detailed Description

All finger views from a single finger minutiae record

G.13.2 Constructor & Destructor Documentation

G.13.2.1 ANSI2004Record() [1/3]

```
BiometricEvaluation::DataInterchange::ANSI2004Record::ANSI2004Record (
            const BE::Memory::uint8Array & fmr,
            const BE::Memory::uint8Array & fir )
```

ANSI2004Record (p. 203) 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. 113) minutia record.
fir	**Finger** (p. 113) image record.

G.13.2.2 ANSI2004Record() [2/3]

```
BiometricEvaluation::DataInterchange::ANSI2004Record::ANSI2004Record (
            const std::string & fmrPath,
            const std::string & firPath )
```

ANSI2004Record (p. 203) 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.

G.13.2.3 ANSI2004Record() [3/3]

```
BiometricEvaluation::DataInterchange::ANSI2004Record::ANSI2004Record (
            const std::initializer_list< BE::Finger::ANSI2004View > & views )
```

ANSI2004Record (p. 203) constructor using a set of finger view records.

Parameters

views	ANSI2004View objects.

G.13.3 Member Function Documentation

G.13.3.1 getEDBLength()

```
uint64_t BiometricEvaluation::DataInterchange::ANSI2004Record::getEDBLength ( ) const  [protected]
```

Obtain the size of EDB that will be written by **getFMR()** (p. 205).

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. 205).

getFMR() (p. 205)

G.13.3.2 getFMR()

```
BE::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.

G.13.3.3 getFMRLength()

```
uint64_t BiometricEvaluation::DataInterchange::ANSI2004Record::getFMRLength ( ) const  [protected]
```

Obtain the size of FMR that will be written by **getFMR()** (p. 205).

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. 205).

getFMR() (p. 205) **getEDBLength()** (p. 205)

G.13.3.4 getMinutia() [1/2]

```
std::vector<BE::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.

G.13.3.5 getMinutia() [2/2]

```
BE::Feature::INCITSMinutiae BiometricEvaluation::DataInterchange::ANSI2004Record::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.

G.13.3.6 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.

G.13.3.7 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. 453)*	viewNumber does not exist.

G.13.3.8 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. 113) view to add.

Returns

View (p. 162) number for view in this record.

G.13.3.9 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. 113) view to add.
viewNumber	**View** (p. 162) number to assign to this view.

Returns

The view number.

Exceptions

BE::Error::StrategyError	viewNumber is not valid.

G.13.3.10 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

BE::Error::ObjectDoesNotExist	viewNumber does not exist.

Note

The remaining view becomes view 1.

G.13.3.11 removeView()

```
void BiometricEvaluation::DataInterchange::ANSI2004Record::removeView (
             const uint64_t viewNumber )
```

Remove a view from the record.

Parameters

viewNumber	The view number to remove.

Exceptions

BE::Error::ObjectDoesNotExist	viewNumber does not exist.

Note

All views will be renumbered after removal.

G.13.3.12 setMinutia() [1/2]

```
void BiometricEvaluation::DataInterchange::ANSI2004Record::setMinutia (
             const std::vector< BE::Feature::INCITSMinutiae > & minutia )
```

Alter the minutia for every finger view.

Parameters

minutia	A vector of INCITSMinutiae for each finger view.

Exceptions

Error::StrategyError *(p. 560)*	Size of minutia does not equal the number of finger views in this record.

G.13.3.13 setMinutia() [2/2]

```
void BiometricEvaluation::DataInterchange::ANSI2004Record::setMinutia (
            uint32_t viewNumber,
            const BE::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. 560)*	**View** (p. 162) number is invalid for this finger record.

G.13.3.14 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. 162) number replaced by view.

Returns

The view number.

Exceptions

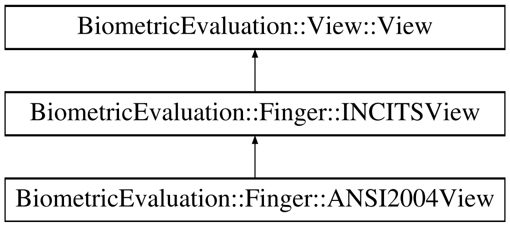
BE::Error::StrategyError	viewNumber is not valid.

G.14 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:


BiometricEvaluation::View::View
BiometricEvaluation::Finger::INCITSView
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 view←Number)

  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.

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.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x20323000

Additional Inherited Members

G.14.1 Detailed Description

A class to represent single finger view and derived information.

A **Finger::ANSI2004View** (p. 211) object represents a finger view from a INCITS/ANSI-2004 **Finger** (p. 113) Minutiae Record.

G.14.2 Constructor & Destructor Documentation

G.14.2.1 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.
in	*firFilename*	The name of the file containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

G.14.2.2 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.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

G.14.3 Member Function Documentation

G.14.3.1 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.

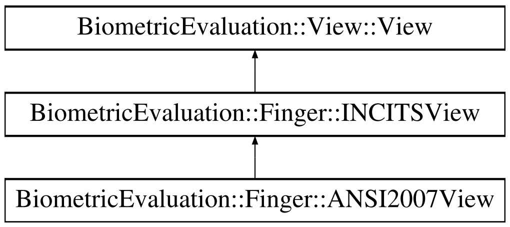
Implements **BiometricEvaluation::Finger::INCITSView** (p. 386).

G.15 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:


BiometricEvaluation::View::View
BiometricEvaluation::Finger::INCITSView
BiometricEvaluation::Finger::ANSI2007View

Public Member Functions

- **ANSI2007View** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view←Number)

  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.

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.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x30333000

Additional Inherited Members

G.15.1 Detailed Description

A class to represent single finger view and derived information.

A **Finger::ANSI2007View** (p. 213) object represents a finger view from a INCITS/ANSI-2007 **Finger** (p. 113) Minutiae Record.

G.15.2 Constructor & Destructor Documentation

G.15.2.1 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.
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. 293)*	Invalid record format.

G.15.2.2 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.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError *(p. 293)*	Invalid record format.

G.15.3 Member Function Documentation

G.15.3.1 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. 386).

G.16 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}

G.16.1 Member Data Documentation

G.16.1.1 cla

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::cla`

The class byte

G.16.1.2 FIELD_LC

`const int BiometricEvaluation::Device::Smartcard::APDU::FIELD_LC {0x00000001} [static]`

Lc field is present; Implies Nc present as well

G.16.1.3 FIELD_LE

`const int BiometricEvaluation::Device::Smartcard::APDU::FIELD_LE {0x00000002} [static]`

Le field is present, response data expected

G.16.1.4 field_mask

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::field_mask`

Mask of optional fields; use field bit masks

G.16.1.5 ins

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::ins`

Instruction byte

G.16.1.6 lc

`uint16_t BiometricEvaluation::Device::Smartcard::APDU::lc`

Lc, length of the Nc field

G.16.1.7 le

`uint16_t BiometricEvaluation::Device::Smartcard::APDU::le`

Le, expected response length

G.16.1.8 nc

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::nc[MAX_NC_SIZE]`

Nc, command data

G.16.1.9 p1

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::p1`

P1 byte

G.16.1.10 p2

`uint8_t BiometricEvaluation::Device::Smartcard::APDU::p2`

P2 byte

G.17 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**

G.17.1 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. 215) that was sent are contained within this object.

G.17.2 Constructor & Destructor Documentation

G.17.2.1 APDUException() [1/2]

```
BiometricEvaluation::Device::Smartcard::APDUException::APDUException ( )  [default]
```

Constructor.

G.17.2.2 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. 215) that was sent.

G.17.3 Member Data Documentation

G.17.3.1 apdu

```
Memory::uint8Array BiometricEvaluation::Device::Smartcard::APDUException::apdu
```

The raw **APDU** (p. 215) that was sent.

G.17.3.2 response

```
APDUResponse BiometricEvaluation::Device::Smartcard::APDUException::response
```

The partial response data and status words from the failed command.

G.18 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**

G.18.1 Detailed Description

The data and status words returned by the card in response to a command.

G.18.2 Constructor & Destructor Documentation

G.18.2.1 APDUResponse() [1/2]

```
BiometricEvaluation::Device::Smartcard::APDUResponse::APDUResponse ( )  [default]
```

Constructor

G.18.2.2 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.

G.18.3 Member Data Documentation

G.18.3.1 data

```
Memory::uint8Array BiometricEvaluation::Device::Smartcard::APDUResponse::data
```

The response data, possibly incomplete

G.18.3.2 sw1

`uint8_t BiometricEvaluation::Device::Smartcard::APDUResponse::sw1 {0}`

status word one

G.18.3.3 sw2

`uint8_t BiometricEvaluation::Device::Smartcard::APDUResponse::sw2 {0}`

status word two

G.19 BiometricEvaluation::Framework::API< T > Class Template Reference

A convenient way to execute biometric technology evaluation **API** (p. 220) 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={}, const bool rethrowExceptions=false)

  *Invoke an operation. 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. 220) operation.*
- std::shared_ptr< BE::Time::Timer > **getTimer** () noexcept

  Obtain the timer object.
- std::shared_ptr< BE::Time::Watchdog > **getWatchdog** () noexcept

  Obtain the watchdog timer object.
- std::shared_ptr< BE::Error::SignalManager > **getSignalManager** () noexcept

  Obtain the signal manager object.

G.19.1 Detailed Description

template<typename T>
class BiometricEvaluation::Framework::API< T >

A convenient way to execute biometric technology evaluation **API** (p. 220) methods safely.

Note

One **API** (p. 220) object should be instantiated per process/thread.

G.19.2 Constructor & Destructor Documentation

G.19.2.1 API()

```
template<typename T >
BiometricEvaluation::Framework::API< T >:: API ( )
```

Constructor

G.19.3 Member Function Documentation

G.19.3.1 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 = {},
            const bool rethrowExceptions = false )
```

Invoke an operation. Invoking operations within this method implicitly wraps the operation in a Signal↩ Manager, Watchdog, and Timer, and follows evaluation best practices for calling an **API** (p. 220) operation.

Parameters

operation	A reference to a function that returns a **Status** (p. 559). (i.e., an **API** (p. 220) method).
success	Operations invoked if operation returns.
failure	Operations invoked if we abort the operation.
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, are rethrown when `rethrowExceptions` is `true`.

Note

success is called and currentState == **APICurrentState::Completed** (p. 116) if operation returns, regardless of the Code of operation's **Status** (p. 559).
Exceptions caught are rethrown after calling failure().

G.19.3.2 getSignalManager()

```
template<typename T >
```

```
std::shared_ptr<BE::Error::SignalManager>  BiometricEvaluation::Framework::API< T >::getSignal↩
Manager ( )  [inline], [noexcept]
```

Obtain the signal manager object.

Returns

Signal manager object.

G.19.3.3 getTimer()

```
template<typename T >
std::shared_ptr<BE::Time::Timer>  BiometricEvaluation::Framework::API< T >::getTimer ( )  [inline],
[noexcept]
```

Obtain the timer object.

Returns

Timer object.

G.19.3.4 getWatchdog()

```
template<typename T >
std::shared_ptr<BE::Time::Watchdog>  BiometricEvaluation::Framework::API< T >::getWatchdog (
)  [inline], [noexcept]
```

Obtain the watchdog timer object.

Returns

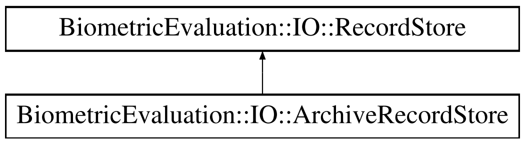
Watchdog timer object.

G.20 BiometricEvaluation::IO::ArchiveRecordStore Class Reference

This class implements the **IO::RecordStore** (p. 500) interface by storing data items in single file, with an associated manifest file.

`#include <be_io_archiverecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ArchiveRecordStore:


BiometricEvaluation::IO::RecordStore
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. 500), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 500), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  *Move the **RecordStore** (p. 500).*
- 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

Static Public Member Functions

- static bool **needsVacuum** (const std::string &pathname)
- static void **vacuum** (const std::string &pathname)

Static Public Attributes

- static const std::string **MANIFEST_FILE_NAME**
- static const std::string **ARCHIVE_FILE_NAME**
- static const long **OFFSET_RECORD_REMOVED** = -1

Additional Inherited Members

G.20.1 Detailed Description

This class implements the **IO::RecordStore** (p. 500) 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.

G.20.2 Constructor & Destructor Documentation

G.20.2.1 ArchiveRecordStore() [1/2]

```
BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore (
             const std::string & pathname,
             const std::string & description )
```

Create a new **ArchiveRecordStore** (p. 222), 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. 454)*	The store already exists.
Error::StrategyError *(p. 560)*	An error occurred when accessing the underlying file system.

G.20.2.2 ArchiveRecordStore() [2/2]

```
BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore (
             const std::string & pathname,
              IO::Mode mode =  IO::Mode::ReadOnly )
```

Open an existing **ArchiveRecordStore** (p. 222).

Parameters

in	*pathname*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	The store does not exist.
Error::StrategyError *(p. 560)*	An error occurred when accessing the underlying file system.

G.20.2.3 ∼ArchiveRecordStore()

```
BiometricEvaluation::IO::ArchiveRecordStore::∼ArchiveRecordStore ( )
```

Destructor.

G.20.3 Member Function Documentation

G.20.3.1 changeDescription()

```
void BiometricEvaluation::IO::ArchiveRecordStore::changeDescription (
             const std::string & description )  [override], [virtual]
```

Change the description of the **RecordStore** (p. 500).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 502).

G.20.3.2 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.20.3.3 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.

G.20.3.4 getCount()

`unsigned int BiometricEvaluation::IO::ArchiveRecordStore::getCount ( ) const [override], [virtual]`

Obtain the number of items in the **RecordStore** (p. 500).

Returns

The number of items in the **RecordStore** (p. 500).

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.20.3.5 getDescription()

`std::string BiometricEvaluation::IO::ArchiveRecordStore::getDescription ( ) const [override], [virtual]`

Obtain a textual description of the **RecordStore** (p. 500).

Returns

The **RecordStore** (p. 500)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.20.3.6 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.

G.20.3.7 getPathname()

`std::string BiometricEvaluation::IO::ArchiveRecordStore::getPathname ( ) const [override], [virtual]`

Return the path name of the **RecordStore** (p. 500).

Returns

Where in the file system the **RecordStore** (p. 500) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.20.3.8 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. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.20.3.9 insert()

```
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. 454)*	A record with the given key is already present.
Error::StrategyError *(p. 560)*	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.20.3.10 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.20.3.11 move()

```
void BiometricEvaluation::IO::ArchiveRecordStore::move (
             const std::string & pathname )  [override], [virtual]
```

Move the **RecordStore** (p. 500).

The **RecordStore** (p. 500) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 507).

G.20.3.12 needsVacuum() [1/2]

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

See if the **ArchiveRecordStore** (p. 222) would benefit from calling **vacuum()** (p. 231) to remove deleted entries, since **vacuum()** (p. 231) is an expensive operation.

Returns

true if **vacuum()** (p. 231) would be beneficial false otherwise

G.20.3.13 needsVacuum() [2/2]

```
static bool BiometricEvaluation::IO::ArchiveRecordStore::needsVacuum (
             const std::string & pathname )  [static]
```

See if the **ArchiveRecordStore** (p. 222) would benefit from calling **vacuum()** (p. 231) to remove deleted entries, since **vacuum()** (p. 231) is an expensive operation.

Parameters

`in`	*pathname*	The path name of the existing **RecordStore** (p. 500).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record with the given key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Returns

true if **vacuum()** (p. 231) would be beneficial false otherwise

G.20.3.14 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 508).

G.20.3.15 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 509).

G.20.3.16 sequence()

```
RecordStore::Record BiometricEvaluation::IO::ArchiveRecordStore::sequence (
            int cursor =  BE_RECSTORE_SEQ_NEXT )  [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.20.3.17 sequenceKey()

```
std::string BiometricEvaluation::IO::ArchiveRecordStore::sequenceKey (
            int cursor =  BE_RECSTORE_SEQ_NEXT )  [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.20.3.18 setCursorAtKey()

```
void BiometricEvaluation::IO::ArchiveRecordStore::setCursorAtKey (
             const std::string & key )  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 500), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 230).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 230).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.20.3.19 sync()

```
void BiometricEvaluation::IO::ArchiveRecordStore::sync ( ) const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.20.3.20 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. 500).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record with the given key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Note

This is an expensive operation.

G.20.4 Member Data Documentation

G.20.4.1 ARCHIVE_FILE_NAME

`const std::string BiometricEvaluation::IO::ArchiveRecordStore::ARCHIVE_FILE_NAME [static]`

Name of the archive file on disk

G.20.4.2 MANIFEST_FILE_NAME

`const std::string BiometricEvaluation::IO::ArchiveRecordStore::MANIFEST_FILE_NAME [static]`

Name of the manifest file on disk

G.20.4.3 OFFSET_RECORD_REMOVED

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

Offset placeholder indicating a removed record

G.21 BiometricEvaluation::Memory::AutoArray< T > Class Template Reference

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

`#include <be_memory_autoarray.h>`

Public Types

- using **value_type** = T
- using **size_type** = size_t
- using **iterator** = **AutoArrayIterator**< false, T >
- using **const_iterator** = **AutoArrayIterator**< true, T >
- using **reference** = T &
- using **const_reference** = const T &

Public Member Functions

- **operator T∗** ()

  *Convert **AutoArray** (p. 232) to T array.*
- **operator const T ∗** () const

  *Convert **AutoArray** (p. 232) 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. 232) with checked access.*
- **const_reference at** (ptrdiff_t index) const

  *Subscript into the **AutoArray** (p. 232) with checked access.*
- **iterator begin** ()

  *Obtain an iterator to the beginning of the **AutoArray** (p. 232).*
- **const_iterator begin** () const

  *Obtain an iterator to the beginning of the **AutoArray** (p. 232).*
- **const_iterator cbegin** () const

  *Obtain an iterator to the beginning of the **AutoArray** (p. 232).*
- **iterator end** ()

  *Obtain an iterator to the end of the **AutoArray** (p. 232).*
- **const_iterator end** () const

  *Obtain an iterator to the end of the **AutoArray** (p. 232).*
- **const_iterator cend** () const

  *Obtain an iterator to the end of the **AutoArray** (p. 232).*
- **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. 232).*
- void **copy** (const T ∗buffer, **size_type size**)

  *Deep-copy the contents of a buffer into this **AutoArray** (p. 232).*
- std::vector< T > **to_vector** () const

  *Obtain a copy of elements in this **AutoArray** (p. 232) as a vector.*
- **AutoArray** (**size_type size**=0)

  *Construct an **AutoArray** (p. 232).*
- **AutoArray** (const **AutoArray** & **copy**)

  *Construct an **AutoArray** (p. 232).*
- **AutoArray** (**AutoArray** &&rvalue) noexcept

  *Construct an **AutoArray** (p. 232).*
- **AutoArray** (std::initializer_list< T > ilist)

  *Construct an **AutoArray** (p. 232).*
- **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** ()

G.21.1 Detailed Description

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

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

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

Forward declaration.

G.21.2 Member Typedef Documentation

G.21.2.1 const_iterator

```
template<class T>
using BiometricEvaluation::Memory::AutoArray< T >:: const_iterator = AutoArrayIterator<true,
T>
```

Const iterator of element

G.21.2.2 const_reference

```
template<class T>
using BiometricEvaluation::Memory::AutoArray< T >:: const_reference = const T&
```

Const reference element

G.21.2.3 iterator

```
template<class T>
using BiometricEvaluation::Memory::AutoArray< T >:: iterator = AutoArrayIterator<false, T>
```

Iterator of element

G.21.2.4 reference

```
template<class T>
using BiometricEvaluation::Memory::AutoArray< T >:: reference = T&
```

Reference to element

G.21.2.5 size_type

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

Type of subscripts, counts, etc.

G.21.2.6 value_type

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

Type of element

G.21.3 Constructor & Destructor Documentation

G.21.3.1 AutoArray() [1/4]

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

Construct an **AutoArray** (p. 232).

Parameters

in	*size*	The number of elements this **AutoArray** (p. 232) should initially hold.

Exceptions

Error::MemoryError *(p. 432)*	Could not allocate new memory.

G.21.3.2 AutoArray() [2/4]

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

Construct an **AutoArray** (p. 232).

Parameters

in	*copy*	An **AutoArray** (p. 232) whose contents will be deep copied into the new **AutoArray** (p. 232).

Exceptions

Error::MemoryError *(p. 432)*	Could not allocate new memory.

G.21.3.3 AutoArray() [3/4]

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

Construct an **AutoArray** (p. 232).

Parameters

in	*rvalue*	An rvalue reference to an **AutoArray** (p. 232) whose contents will be moved and destroyed.

G.21.3.4 AutoArray() [4/4]

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

Construct an **AutoArray** (p. 232).

Parameters

in	*ilist*	An initializer list of type T.

G.21.3.5 ∼AutoArray()

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

Destructor

G.21.4 Member Function Documentation

G.21.4.1 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. 232) 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. 232).

G.21.4.2 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. 232) 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. 232).

G.21.4.3 begin() [1/2]

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T >::  iterator  BiometricEvaluation::Memory::Auto↩
Array< T >::begin ( )
```

Obtain an iterator to the beginning of the **AutoArray** (p. 232).

Returns

Iterator positioned at the first element of the **AutoArray** (p. 232).

G.21.4.4 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. 232).

Returns

Const iterator positioned at the first element of the **AutoArray** (p. 232).

G.21.4.5 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. 232).

Returns

Const iterator positioned at the first element of the **AutoArray** (p. 232).

G.21.4.6 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. 232).

Returns

Iterator positioned at the one-past-last element of the **AutoArray** (p. 232).

G.21.4.7 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. 232).

Parameters

in	*buffer*	An allocated buffer whose contents will be deep-copied into this object. Only **size()** (p. 241) bytes will be copied.

Warning

If buffer is smaller in size than the current size of the **AutoArray** (p. 232), you MUST call **copy(const T∗, size_type)** (p. 238). This method must only be used when buffer is larger than or equal to the size of the **AutoArray** (p. 232).

G.21.4.8 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. 232).

Parameters

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

Warning

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

G.21.4.9 end() [1/2]

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T >::  iterator  BiometricEvaluation::Memory::Auto↩
Array< T >::end ( )
```

Obtain an iterator to the end of the **AutoArray** (p. 232).

Returns

Iterator positioned at the one-past-last element of the **AutoArray** (p. 232).

G.21.4.10 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. 232).

Returns

Iterator positioned at the one-past-last element of the **AutoArray** (p. 232).

G.21.4.11 operator const T *()

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

Convert **AutoArray** (p. 232) to const T array.

Returns

Const pointer to the beginning of the underlying array storage.

G.21.4.12 operator T*()

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

Convert **AutoArray** (p. 232) to T array.

Returns

Pointer to the beginning of the underlying array storage.

G.21.4.13 operator=() [1/2]

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T > & BiometricEvaluation::Memory::AutoArray< T >↩
::operator= (
             const AutoArray< T > & other )
```

Copy assignment operator overload performing a deep copy.

Parameters

in	*other*	**AutoArray** (p. 232) to be copied.

Returns

Reference to a new **AutoArray** (p. 232) object, the lvalue **AutoArray** (p. 232).

Exceptions

Error::MemoryError *(p. 432)*	Could not allocate new memory.

G.21.4.14 operator=() [2/2]

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

Move assignment operator.

Parameters

in	*other*	rvalue reference to another **AutoArray** (p. 232), whose contents will be moved and cleared from itself.

Returns

Reference to the lvalue **AutoArray** (p. 232).

G.21.4.15 operator[]() [1/2]

```
template<class T >
BiometricEvaluation::Memory::AutoArray< T >:: reference BiometricEvaluation::Memory::Auto↩
Array< 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.

G.21.4.16 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.

G.21.4.17 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. 232) 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. 432)*	Problem allocating memory.

G.21.4.18 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. 241) has been called, the value returned from **size()** (p. 241) may be smaller than the actual allocated size of the underlying storage.

G.21.4.19 to_vector()

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

Obtain a copy of elements in this **AutoArray** (p. 232) 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. 232) where not all elements have been initialized will likely cause undefined behavior.

Returns

A vector containing the contents of this **AutoArray** (p. 232).

G.22 BiometricEvaluation::Memory::AutoArrayIterator< CONST, T > Class Template Reference

RandomAccessIterator for any **AutoArray** (p. 232).

```
#include <be_memory_autoarrayiterator.h>
```

Public Types

- using **iterator_category** = std::random_access_iterator_tag
- using **value_type** = typename std::conditional< CONST, const T, T >::type
- using **difference_type** = std::ptrdiff_t
- using **pointer** = typename std::conditional< CONST, const T ∗, T ∗ >::type
- using **reference** = typename std::conditional< CONST, const T &, T & >::type
- using **container** = typename std::conditional< CONST, const **AutoArray**< T > ∗, **AutoArray**< T > ∗ >::type

  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 postfix)
- **AutoArrayIterator operator--** (int postfix)
- **AutoArrayIterator operator+** (const **AutoArrayIterator** &rhs) const
- **difference_type operator-** (const **AutoArrayIterator**< CONST, 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)

G.22.1 Detailed Description

template<bool CONST, class T>
class BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >

RandomAccessIterator for any **AutoArray** (p. 232).

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. 232).

G.22.2 Member Typedef Documentation

G.22.2.1 difference_type

```
template<bool CONST, class T>
using  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::  difference_type = std↩
::ptrdiff_t
```

Type used to measure distance between iterators

G.22.2.2 iterator_category

```
template<bool CONST, class T>
using  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::  iterator_category = std↩
::random_access_iterator_tag
```

Type of iterator

G.22.2.3 pointer

```
template<bool CONST, class T>
using  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::  pointer = typename std↩
::conditional<CONST, const T*, T*>::type
```

Pointer to the type iterated over

G.22.2.4 reference

```
template<bool CONST, class T>
using  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::  reference = typename std↩
::conditional<CONST, const T&, T&>::type
```

Reference to the type iterated over

G.22.2.5 value_type

```
template<bool CONST, class T>
using  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::  value_type = typename std↩
::conditional<CONST, const T, T>::type
```

Type when dereferencing iterators

G.22.3 Constructor & Destructor Documentation

G.22.3.1 AutoArrayIterator() [1/3]

```
template<bool CONST, class T>
BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::  AutoArrayIterator (
              container autoArray = nullptr,
              difference_type offset = 0 )  [inline]
```

Default constructor.

Parameters

autoArray	Pointer to the **AutoArray** (p. 232) to iterate
offset	The offset into the **AutoArray** (p. 232) where this iterator should start.

G.22.3.2 AutoArrayIterator() [2/3]

```
template<bool CONST, class T>
BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >:: AutoArrayIterator (
            const AutoArrayIterator< CONST, T > & rhs ) [default]
```

Default copy constructor

G.22.3.3 AutoArrayIterator() [3/3]

```
template<bool CONST, class T>
BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >:: AutoArrayIterator (
             AutoArrayIterator< CONST, T > && rhs ) [default]
```

Default move constructor

G.22.3.4 ∼AutoArrayIterator()

```
template<bool CONST, class T>
BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::∼ AutoArrayIterator ( ) [default]
```

Default destructor

G.22.4 Member Function Documentation

G.22.4.1 operator"!=()

```
template<bool CONST, class T>
bool BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator!= (
            const AutoArrayIterator< CONST, T > & rhs ) const [inline]
```

Returns

Whether or not the offsets are different.

G.22.4.2 operator∗()

```
template<bool CONST, class T>
reference BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator∗ ( ) const [inline]
```

Returns

Object at the current offset.

G.22.4.3 operator+() [1/2]

```
template<bool CONST, class T>
AutoArrayIterator BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator+ (
            const AutoArrayIterator< CONST, T > & rhs ) const [inline]
```

Returns

This object with offset incremented by rhs' offset.

G.22.4.4 operator+() [2/2]

```
template<bool CONST, class T>
AutoArrayIterator BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator+ (
            const difference_type & rhs ) const [inline]
```

Returns

This object with offset incremented rhs.

G.22.4.5 operator++() [1/2]

```
template<bool CONST, class T>
AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator++ (
) [inline]
```

Returns

This object with incremented offset.

G.22.4.6 operator++() [2/2]

```
template<bool CONST, class T>
AutoArrayIterator BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator++ (
            int postfix ) [inline]
```

Returns

This object before incrementing offset.

G.22.4.7 operator+=()

```
template<bool CONST, class T>
AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator+= (
            const difference_type & rhs ) [inline]
```

Returns

This object with rhs added to offset.

G.22.4.8 operator-() [1/2]

```
template<bool CONST, class T>
difference_type BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator- (
            const AutoArrayIterator< CONST, T > & rhs ) const [inline]
```

Returns

Offset decremented by rhs' offset.

G.22.4.9 operator-() [2/2]

```
template<bool CONST, class T>
AutoArrayIterator  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator- (
            const  difference_type & rhs ) const  [inline]
```

Returns

This object with offset decremented rhs.

G.22.4.10 operator--() [1/2]

```
template<bool CONST, class T>
AutoArrayIterator&  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator-- (
)  [inline]
```

Returns

This object with decremented offset.

G.22.4.11 operator--() [2/2]

```
template<bool CONST, class T>
AutoArrayIterator  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator-- (
            int postfix )  [inline]
```

Returns

This object before decrementing offset.

G.22.4.12 operator-=()

```
template<bool CONST, class T>
AutoArrayIterator&  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator-= (
            const  difference_type & rhs )  [inline]
```

Returns

This object with rhs removed from offset.

G.22.4.13 operator->()

```
template<bool CONST, class T>
pointer  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator-> ( ) const  [inline]
```

Returns

Address of object at the current offset.

G.22.4.14 operator<()

```
template<bool CONST, class T>
bool BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator< (
            const AutoArrayIterator< CONST, T > & rhs ) const [inline]
```

Returns

true if this offset is < rhs'.

G.22.4.15 operator<=()

```
template<bool CONST, class T>
bool BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator<= (
            const AutoArrayIterator< CONST, T > & rhs ) const [inline]
```

Returns

true if this offset is <= rhs'.

G.22.4.16 operator=() [1/2]

```
template<bool CONST, class T>
AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator= (
             pointer rhs ) [inline]
```

Returns

This object with offset set to rhs.

G.22.4.17 operator=() [2/2]

```
template<bool CONST, class T>
AutoArrayIterator& BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator= (
            const AutoArrayIterator< CONST, T > & rhs ) [inline], [default]
```

Default assignment operator.

G.22.4.18 operator==()

```
template<bool CONST, class T>
bool BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator== (
            const AutoArrayIterator< CONST, T > & rhs ) const [inline]
```

Returns

Whether or not the offsets are the same.

G.22.4.19 operator>()

```
template<bool CONST, class T>
bool  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator> (
            const  AutoArrayIterator< CONST, T > & rhs ) const  [inline]
```

Returns

true if this offset is > rhs'.

G.22.4.20 operator>=()

```
template<bool CONST, class T>
bool  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator>= (
            const  AutoArrayIterator< CONST, T > & rhs ) const  [inline]
```

Returns

true if this offset is >= rhs'.

G.22.4.21 operator[]()

```
template<bool CONST, class T>
reference  BiometricEvaluation::Memory::AutoArrayIterator< CONST, T >::operator[] (
            const  difference_type & rhs ) const  [inline]
```

Returns

Object at rhs.

G.22.5 Friends And Related Function Documentation

G.22.5.1 operator+

```
template<bool CONST, class T>
AutoArrayIterator operator+ (
            const  difference_type & lhs,
            const  AutoArrayIterator< CONST, T > & rhs )  [friend]
```

Returns

New iterator combining offsets.

G.22.5.2 operator-

```
template<bool CONST, class T>
AutoArrayIterator operator- (
            const  difference_type & lhs,
            const  AutoArrayIterator< CONST, T > & rhs )  [friend]
```

Returns

New iterator differing offsets, iterating rhs' **AutoArray** (p. 232).

G.23 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)

G.23.1 Member Typedef Documentation

G.23.1.1 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. 250) as a wrapper for a pointer rather than the object it truly is. Therefore, you can interact with the **AutoBuffer** (p. 250) 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. 250) object like:

```
AutoBuffer<ANSI_NIST> obj = AutoBuffer(allocator_fn,
    deallocator_fn[, copy_constructor]);
```

Notice the **AutoBuffer** (p. 250) is for ANSI_NIST and not ANSI_NIST∗, since **AutoBuffer** (p. 250) will handle the pointer for you. You can pass the **AutoBuffer<ANSI_NIST>** (p. 250) 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∗:

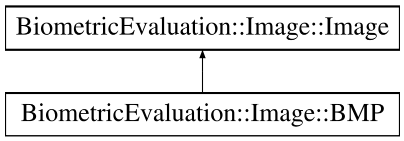
```
int size = obj->num_bytes;
```

G.24 BiometricEvaluation::Image::BMP Class Reference

A BMP-encoded image.

`#include <be_image_bmp.h>`

Inheritance diagram for BiometricEvaluation::Image::BMP:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::BMP

Classes

- struct **ColorTableEntry**

Public Types

- using **ColorTableEntry** = struct **ColorTableEntry**
- using **ColorTable** = std::vector< **ColorTableEntry** >

Public Member Functions

- **BMP** (const uint8_t *data, const uint64_t size)
- **BMP** (const **Memory::uint8Array** &data)
- **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.

Static Public Member Functions

- static bool **isBMP** (const uint8_t *data, uint64_t size)

Additional Inherited Members

G.24.1 Detailed Description

A BMP-encoded image.

Note

Only supports uncompressed BMPs with the 40-byte BITMAPINFOHEADER header information with no compression or RLE8 compression.

G.24.2 Member Function Documentation

G.24.2.1 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.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 357).

G.24.2.2 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. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 470)*	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. 358).

G.24.2.3 isBMP()

```
static bool BiometricEvaluation::Image::BMP::isBMP (
            const uint8_t * data,
            uint64_t size )  [static]
```

Whether or not data is a **BMP** (p. 251) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a **BMP** (p. 251) image, false otherwise.

G.25 BiometricEvaluation::DataInterchange::AN2KRecord::Character↩ Set Struct Reference

Public Member Functions

- **CharacterSet** (uint16_t **identifier**=0, std::string **commonName**="", std::string **version**="")
  *Create a new **CharacterSet** (p. 253) struct.*

Public Attributes

- uint16_t **identifier**
- std::string **commonName**
- std::string **version**

G.25.1 Constructor & Destructor Documentation

G.25.1.1 CharacterSet()

```
BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::CharacterSet (
            uint16_t identifier = 0,
            std::string commonName = "",
            std::string version = "" )  [inline]
```

Create a new **CharacterSet** (p. 253) struct.

Parameters

identifier	Numeric identifier of the character set.
commonName	Common name of the character set.
version	Optional version number of the character set.

G.25.2 Member Data Documentation

G.25.2.1 commonName

`std::string BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::commonName`

Common name of the character set

G.25.2.2 identifier

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

Identifier (000-999)

G.25.2.3 version

`std::string BiometricEvaluation::DataInterchange::AN2KRecord::CharacterSet::version`

Optional version of the character set

G.26 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**

G.26.1 Detailed Description

One element of the colormap table.

G.26.2 Member Data Documentation

G.26.2.1 blue

`uint8_t BiometricEvaluation::Image::BMP::ColorTableEntry::blue`

Blue value

G.26.2.2 green

`uint8_t BiometricEvaluation::Image::BMP::ColorTableEntry::green`

Green value

G.26.2.3 red

`uint8_t BiometricEvaluation::Image::BMP::ColorTableEntry::red`

Red value

G.26.2.4 reserved

```
uint8_t BiometricEvaluation::Image::BMP::ColorTableEntry::reserved
```

Reserved value

G.27 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**

G.27.1 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.

G.27.2 Member Data Documentation

G.27.2.1 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).

G.27.2.2 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.

G.27.2.3 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.

G.28 BiometricEvaluation::Process::CommandCenter< T, typename > Class Template Reference

```
#include <be_process_commandcenter.h>
```

Classes

- class **Command**

Public Member Functions

- **CommandCenter** (uint16_t port= **MessageCenter::DEFAULT_PORT**)

  Constructor.
- ∼**CommandCenter** ()=default
- bool **hasPendingCommands** ()

  Determine if there are commands waiting.
- bool **getNextCommand** (**Command** &command, int numSeconds=-1, std::string invalidCommand←Response="")

  Get the next command.
- void **sendResponse** (uint32_t clientID, const std::string &response, const std::string prefix=">> ", const std::string suffix="\)

  Send a string response to a client.
- void **disconnectClient** (uint32_t clientID)

  Break the connection with a client.

G.28.1 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.

G.28.2 Constructor & Destructor Documentation

G.28.2.1 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.

G.28.2.2 ∼CommandCenter()

```
template<typename T, typename = typename std::enable_if<std::is_enum<T>::value>>
BiometricEvaluation::Process::CommandCenter< T, typename >::∼ CommandCenter ( ) [default]
```

Destructor (default).

G.28.3 Member Function Documentation

G.28.3.1 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.

G.28.3.2 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. 255) that will be populated when this method returns true.
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.

G.28.3.3 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. 255):: **getNextCommand()** (p. 257)

G.28.3.4 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.

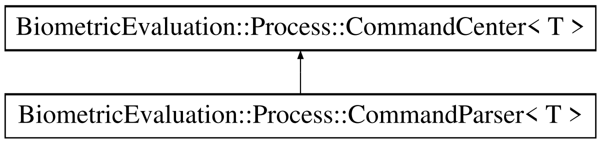
Parameters

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.

G.29 BiometricEvaluation::Process::CommandParser< T > Class Template Reference

`#include <be_process_commandcenter.h>`

Inheritance diagram for BiometricEvaluation::Process::CommandParser< T >:


BiometricEvaluation::Process::CommandCenter< T >
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

G.29.1 Detailed Description

template<typename T>
class BiometricEvaluation::Process::CommandParser< T >

Abstraction to parse messages received via **CommandCenter** (p. 255).

G.29.2 Constructor & Destructor Documentation

G.29.2.1 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.

G.29.2.2 ∼CommandParser()

```
template<typename T >
virtual  BiometricEvaluation::Process::CommandParser< T >::∼ CommandParser ( )  [virtual], [default]
```

Virtual destructor (default).

G.29.3 Member Function Documentation

G.29.3.1 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.
numSeconds	Number of seconds to wait for a command, or -1 to block indefinitely.

Returns

true if command has been populated, false otherwise.

G.29.3.2 getUsage()

```
template<typename T >
std::string BiometricEvaluation::Process::CommandParser< T >::getUsage ( ) const [inline]
```

Returns

Usage string.

G.29.3.3 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.

G.29.3.4 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

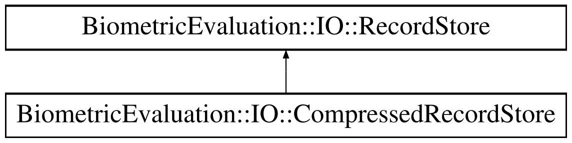
If not set, no additional usage is sent.

G.30 BiometricEvaluation::IO::CompressedRecordStore Class Reference

Sibling-implemented **RecordStore** (p. 500) with Compression.

`#include <be_io_compressedrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::CompressedRecordStore:


BiometricEvaluation::IO::RecordStore
BiometricEvaluation::IO::CompressedRecordStore

Public Member Functions

- **CompressedRecordStore** (const std::string &pathname, const std::string &description, const **Record**↩ **Store::Kind** &recordStoreType, const std::string &compressorType)

- **CompressedRecordStore** (const std::string &pathname, const std::string &description, const **Record**↩ **Store::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. 500), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  Sequence through a ***RecordStore*** *(p. 500), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  Move the ***RecordStore*** *(p. 500).*
- **CompressedRecordStore** (const **CompressedRecordStore** &rhs)=delete

  Copy constructor (disabled).
- **CompressedRecordStore** & **operator=** (const **CompressedRecordStore** &rhs)=delete

  Assignment operator (disabled).

Additional Inherited Members

G.30.1 Detailed Description

Sibling-implemented **RecordStore** (p. 500) with Compression.

G.30.2 Constructor & Destructor Documentation

G.30.2.1 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. 260), read/write mode.

Parameters

in	*pathname*	The directory where the store is to be created.

Parameters

in	*description*	The store's description.
in	*recordStoreType*	The type of **RecordStore** (p. 500) subclass the internal RecordStores should be.
in	*compressorType*	The type of compression that should be used within the internal RecordStores.

Exceptions

Error::ObjectExists *(p. 454)*	The store already exists.
Error::StrategyError *(p. 560)*	An error occurred when accessing the underlying file system.

G.30.2.2 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. 260), 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. 500) subclass the internal RecordStores should be.
in	*compressorType*	The type of compression that should be used within the internal RecordStores.

Exceptions

Error::ObjectExists *(p. 454)*	The store already exists.
Error::StrategyError *(p. 560)*	An error occurred when accessing the underlying file system.

G.30.2.3 CompressedRecordStore() [3/4]

```
BiometricEvaluation::IO::CompressedRecordStore::CompressedRecordStore (
             const std::string & pathname,
              IO::Mode mode =  IO::Mode::ReadOnly )
```

Open an existing **CompressedRecordStore** (p. 260).

Parameters

in	*pathname*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotExist (p. *453*)	The store does not exist.
Error::StrategyError (p. *560*)	An error occurred when accessing the underlying file system.

G.30.2.4 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. 260) object to copy.

G.30.3 Member Function Documentation

G.30.3.1 changeDescription()

```
void BiometricEvaluation::IO::CompressedRecordStore::changeDescription (
             const std::string & description )  [override], [virtual]
```

Change the description of the **RecordStore** (p. 500).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 502).

G.30.3.2 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.30.3.3 getCount()

`unsigned int BiometricEvaluation::IO::CompressedRecordStore::getCount ( ) const [override], [virtual]`

Obtain the number of items in the **RecordStore** (p. 500).

Returns

The number of items in the **RecordStore** (p. 500).

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.30.3.4 getDescription()

`std::string BiometricEvaluation::IO::CompressedRecordStore::getDescription ( ) const [override], [virtual]`

Obtain a textual description of the **RecordStore** (p. 500).

Returns

The **RecordStore** (p. 500)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.30.3.5 getPathname()

`std::string BiometricEvaluation::IO::CompressedRecordStore::getPathname ( ) const [override], [virtual]`

Return the path name of the **RecordStore** (p. 500).

Returns

Where in the file system the **RecordStore** (p. 500) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.30.3.6 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. 500).

Exceptions

Error::StrategyError (p. 560)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.30.3.7 insert()

```
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. 454)	A record with the given key is already present.
Error::StrategyError (p. 560)	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.30.3.8 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. 453)	A record for the key does not exist.
Error::StrategyError (p. 560)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.30.3.9 move()

```
void BiometricEvaluation::IO::CompressedRecordStore::move (
            const std::string & pathname )  [override], [virtual]
```

Move the **RecordStore** (p. 500).

The **RecordStore** (p. 500) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 507).

G.30.3.10 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. 260) object to assign.

Returns

CompressedRecordStore (p. 260) object, now containing the contents of rhs.

G.30.3.11 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 508).

G.30.3.12 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 509).

G.30.3.13 sequence()

```
RecordStore::Record BiometricEvaluation::IO::CompressedRecordStore::sequence (
            int cursor = BE_RECSTORE_SEQ_NEXT )  [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.30.3.14 sequenceKey()

```
std::string BiometricEvaluation::IO::CompressedRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT )  [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.30.3.15 setCursorAtKey()

```
void BiometricEvaluation::IO::CompressedRecordStore::setCursorAtKey (
            const std::string & key )  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 500), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 267).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 267).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.30.3.16 sync()

```
void BiometricEvaluation::IO::CompressedRecordStore::sync ( ) const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

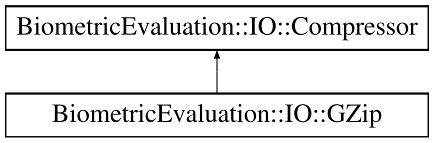
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.31 BiometricEvaluation::IO::Compressor Class Reference

`#include <be_io_compressor.h>`

Inheritance diagram for BiometricEvaluation::IO::Compressor:


BiometricEvaluation::IO::Compressor
BiometricEvaluation::IO::GZip

Public Types

- enum **Kind** { **GZIP** }

Public Member Functions

- **Compressor** ()

  Create a new ***Compressor*** *(p. 269) object.*
- virtual **Memory::uint8Array compress** (const uint8_t ∗const uncompressedData, uint64_t uncompressed↩ DataSize) const =0

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

  Compress a buffer.
- virtual void **compress** (const uint8_t ∗const uncompressedData, uint64_t uncompressedDataSize, const std::string &outputFile) const =0

  Compress a buffer.
- virtual void **compress** (const **Memory::uint8Array** &uncompressedData, const std::string &output↩ File) const =0

  Compress a buffer.

- virtual **Memory::uint8Array** **compress** (const std::string &inputFile) const =0

  Compress a file.
- virtual void **compress** (const std::string &inputFile, const std::string &outputFile) const =0

  Compress a file.
- virtual **Memory::uint8Array** **decompress** (const uint8_t ∗const compressedData, uint64_t compressed↩ DataSize) const =0

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

  Decompress a compressed buffer.
- virtual **Memory::uint8Array** **decompress** (const std::string &inputFile) const =0

  Decompress a compressed buffer into a file.
- virtual void **decompress** (const **Memory::uint8Array** &compressedData, const std::string &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)

G.31.1 Detailed Description

Implementations for compressing and decompressing data

G.31.2 Member Enumeration Documentation

G.31.2.1 Kind

```
enum BiometricEvaluation::IO::Compressor::Kind [strong]
```

Kinds of Compressors (for factory)

G.31.3 Constructor & Destructor Documentation

G.31.3.1 Compressor() [1/2]

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

Create a new **Compressor** (p. 269) object.
Default compression options will be used.

G.31.3.2 ∼Compressor()

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

Destructor

G.31.3.3 Compressor() [2/2]

```
BiometricEvaluation::IO::Compressor::Compressor (
             const Compressor & other ) [delete]
```

Copy constructor (disabled).
Disabled because **Properties** (p. 482) member cannot be copied.

Parameters

other	**Compressor** (p. 269) to copy.

G.31.4 Member Function Documentation

G.31.4.1 compress() [1/6]

```
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 uncompressedData.

Returns

Compressed buffer.

Exceptions

Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 345).

G.31.4.2 compress() [2/6]

```
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. 560)*	**Error** (p. 106) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 345).

G.31.4.3 compress() [3/6]

```
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 uncompressedData.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists *(p. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 345).

G.31.4.4 compress() [4/6]

```
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. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 346).

G.31.4.5 compress() [5/6]

```
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. 453)*	Input file does not exist.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 346).

G.31.4.6 compress() [6/6]

```
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. 453)*	Input file does not exist.
Error::ObjectExists *(p. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 347).

G.31.4.7 createCompressor()

```
static std::shared_ptr< Compressor> BiometricEvaluation::IO::Compressor::createCompressor (
             Compressor::Kind compressorKind = Kind::GZIP )  [static]
```

Compressor (p. 269) factory.

Parameters

compressorKind	A known kind of compressor.

Returns

A new compressor with default options.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	Invalid compressor type.

G.31.4.8 decompress() [1/6]

```
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.

Parameters

compressedDataSize	Size of compressedData.

Returns

Decompressed data.

Exceptions

Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 347).

G.31.4.9 decompress() [2/6]

```
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. 560)*	**Error** (p. 106) in decompression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 348).

G.31.4.10 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. 560)*	**Error** (p. 106) in decompression unit.
Error::ObjectDoesNotExists	Output file already exists.

Implemented in **BiometricEvaluation::IO::GZip** (p. 348).

G.31.4.11 decompress() [4/6]

```
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. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 350).

G.31.4.12 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.
compressedDataSize	Size of compressedData.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists *(p. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 349).

G.31.4.13 decompress() [6/6]

```
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. 453)*	Input file does not exist.
Error::ObjectExists *(p. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implemented in **BiometricEvaluation::IO::GZip** (p. 349).

G.31.4.14 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.

G.31.4.15 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. 453)*	The option was never set.

G.31.4.16 operator=()

```
Compressor& BiometricEvaluation::IO::Compressor::operator= (
            const  Compressor & other )  [delete]
```

Assignment overload (disabled).

Disabled because **Properties** (p. 482) member cannot be assigned.

Parameters

other	**Compressor** (p. 269) to assign.

Returns

lhs **Compressor** (p. 269).

G.31.4.17 removeOption()

```
void BiometricEvaluation::IO::Compressor::removeOption (
            const std::string & optionName )
```

Remove a compressor option.

Parameters

optionName	Name of the option to remove.

G.31.4.18 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. 560)*	**Error** (p. 106) setting option.

G.31.4.19 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. 560)*	**Error** (p. 106) setting option.

G.32 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. 280) from a memory buffer.*
- **Container** (const std::shared_ptr< **Memory::uint8Array** > &buffer)

  *Construct a **Container** (p. 280) from a memory buffer wrapped in a shared pointer.*
- **Container** (const std::string &filename)

  *Construct a **Container** (p. 280) 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. *162*) *streams are indexed independently from other streams in the container.*

G.32.1 Detailed Description

Representation of a video container.

The **Container** (p. 280) class represents a single container stream that can be used to access the video and audio components of the stream.

G.32.2 Constructor & Destructor Documentation

G.32.2.1 Container() [1/3]

```
BiometricEvaluation::Video::Container::Container (
              const  Memory::uint8Array & buffer )
```

Construct a **Container** (p. 280) from a memory buffer.

Using this constructor can result in buffer memory usage twice that of other constructors.

Exceptions

Error::MemoryError (p. *432*)	**Error** (p. 106) allocating memory for internal buffering.
Error::StrategyError (p. *560*)	Other error when reading the container stream.

G.32.2.2 Container() [2/3]

```
BiometricEvaluation::Video::Container::Container (
              const std::shared_ptr<  Memory::uint8Array > & buffer )
```

Construct a **Container** (p. 280) from a memory buffer wrapped in a shared pointer.

Applications must not modify the data underlying the AutoArray.

Exceptions

Error::MemoryError (p. *432*)	**Error** (p. 106) allocating memory for internal buffering.
Error::StrategyError (p. *560*)	Other error when reading the container stream.

G.32.2.3 Container() [3/3]

```
BiometricEvaluation::Video::Container::Container (
              const std::string & filename )
```

Construct a **Container** (p. 280) from file.

Exceptions

Error::ObjectDoesNotExist (p. *453*)	File does not exist.

Exceptions

Error::MemoryError *(p. 432)*	**Error** (p. 106) allocating memory for internal buffering.
Error::StrategyError *(p. 560)*	Other error when reading the container stream.

G.32.3 Member Function Documentation

G.32.3.1 getVideoStream()

```
std::unique_ptr< Video::Stream> BiometricEvaluation::Video::Container::getVideoStream (
            uint32_t videoNum )
```

Obtain a video stream from the container. **Video** (p. 162) streams are indexed independently from other streams in the container.

Parameters

videoNum	The number of the video stream within the container.

Exceptions

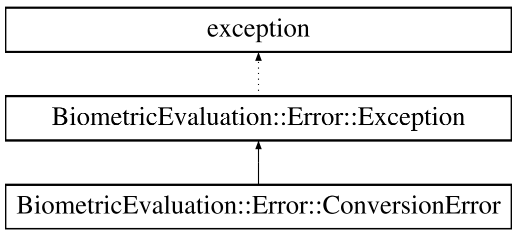
Error::ParameterError *(p. 470)*	The requested video stream is not available.

G.33 BiometricEvaluation::Error::ConversionError Class Reference

Error (p. 106) 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:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ConversionError

Public Member Functions

- **ConversionError** ()
- **ConversionError** (const std::string &info)

G.33.1 Detailed Description

Error (p. 106) when converting one object into another, a property value from string to int, for example.

G.33.2 Constructor & Destructor Documentation

G.33.2.1 ConversionError() [1/2]

```
BiometricEvaluation::Error::ConversionError::ConversionError ( )
```

Construct a **ConversionError** (p. 282) object with the default information string.

G.33.2.2 ConversionError() [2/2]

```
BiometricEvaluation::Error::ConversionError::ConversionError (
            const std::string & info )
```

Construct a **ConversionError** (p. 282) object with an information string appended to the default information string.

G.34 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. 283) struct.*

Public Attributes

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

G.34.1 Detailed Description

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

G.34.2 Constructor & Destructor Documentation

G.34.2.1 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. 283) struct.

Parameters

in	*x*	X-coordinate

Parameters

in	*y*	Y-coordinate
in	*xDistance*	X-coordinate distance from origin
in	*yDistance*	Y-coordinate distance from origin

G.34.3 Member Data Documentation

G.34.3.1 x

`uint32_t BiometricEvaluation::Image::Coordinate::x`

X-coordinate

G.34.3.2 xDistance

`float BiometricEvaluation::Image::Coordinate::xDistance`

X-coordinate distance from origin

G.34.3.3 y

`uint32_t BiometricEvaluation::Image::Coordinate::y`

Y-coordinate

G.34.3.4 yDistance

`float BiometricEvaluation::Image::Coordinate::yDistance`

Y-coordinate distance from origin

G.35 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**

G.36 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. 284) struct.*

Public Attributes

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

G.36.1 Detailed Description

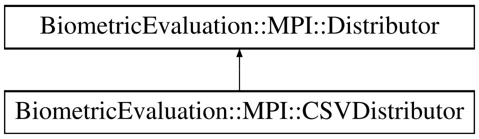
Representation of the core.

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.

G.37 BiometricEvaluation::MPI::CSVDistributor Class Reference

`#include <be_mpi_csvdistributor.h>`

Inheritance diagram for BiometricEvaluation::MPI::CSVDistributor:


BiometricEvaluation::MPI::Distributor
BiometricEvaluation::MPI::CSVDistributor

Public Member Functions

- **CSVDistributor** (const std::string &propertiesFileName, const std::string &delimiter="")

  *Construct a **CSVDistributor** (p. 285) using named properties.*

Protected Member Functions

- void **createWorkPackage** (**MPI::WorkPackage** &workPackage)

  Create a work package for distribution.

G.37.1 Detailed Description

Distribute lines of a text file via Work Packages

G.37.2 Constructor & Destructor Documentation

G.37.2.1 CSVDistributor()

```
BiometricEvaluation::MPI::CSVDistributor::CSVDistributor (
             const std::string & propertiesFileName,
             const std::string & delimiter = "" )
```

Construct a **CSVDistributor** (p. 285) using named properties.

Parameters

in	*propertiesFileName*	The file containing the properties.
in	*delimiter*	Delimiter used to tokenize lines read from CSV.

G.37.3 Member Function Documentation

G.37.3.1 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.

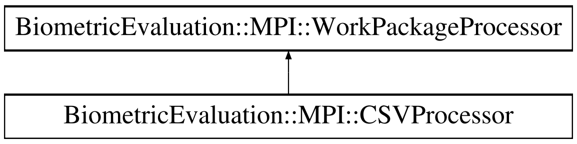
Implements **BiometricEvaluation::MPI::Distributor** (p. 304).

G.38 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:


BiometricEvaluation::MPI::WorkPackageProcessor
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. *148*) *the data contents of the work package. This method is part of the worker personality.*

Protected Member Functions

- std::shared_ptr< **MPI::CSVResources** > **getResources** ()

G.38.1 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.

G.38.2 Constructor & Destructor Documentation

G.38.2.1 CSVProcessor()

```
BiometricEvaluation::MPI::CSVProcessor::CSVProcessor (
            const std::string & propertiesFileName )
```

Construct a work package processor with the given properties.

A **CSVProcessor** (p. 286) 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. 307)*	An error occurred, usually due to missing or incorrect properties.

G.38.3 Member Function Documentation

G.38.3.1 newProcessor()

```
virtual std::shared_ptr< WorkPackageProcessor> BiometricEvaluation::MPI::CSVProcessor::new←
Processor (
            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. 416) 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. 602). If an error occurs during construction, throw a **Error::Exception** (p. 307) with a message to be caught and logged.

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 603).

G.38.3.2 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. 416) that may be used to save messages generated by the object.

Exceptions

Error::Exception *(p. 307)*	An implementation specific error occurred. The exception string will be logged by the **Framework** (p. 115).

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 603).

G.38.3.3 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).
in	*line*	The key associated with the record that is to be processed.

Exceptions

Error::Exception *(p. 307)*	An error occurred processing the record: Missing record, input/output error, or memory allocation.

G.38.3.4 processWorkPackage()

```
void BiometricEvaluation::MPI::CSVProcessor::processWorkPackage (
            MPI::WorkPackage & workPackage )  [virtual]
```

Process (p. 148) the data contents of the work package. This method is part of the worker personality.

Parameters

in	*workPackage*	The work package.

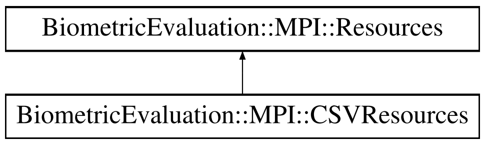
Exceptions

Error::Exception *(p. 307)*	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 604).

G.39 BiometricEvaluation::MPI::CSVResources Class Reference

Inheritance diagram for BiometricEvaluation::MPI::CSVResources:


BiometricEvaluation::MPI::Resources
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. 291) by a* ***Distributor*** *(p. 303).*
- 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.

Static Public Member Functions

- static std::vector< std::string > **getRequiredProperties** ()
- static std::vector< std::string > **getOptionalProperties** ()

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**

G.39.1 Member Function Documentation

G.39.1.1 getDelimiter()

```
std::string BiometricEvaluation::MPI::CSVResources::getDelimiter ( ) const
```

Returns

Delimiter used to tokenize sent lines.

G.39.1.2 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. 560)	Neither CSV file open nor CSV buffer populated.

G.39.1.3 getNumRemainingLines()

`uint64_t BiometricEvaluation::MPI::CSVResources::getNumRemainingLines ( ) const`

Obtain the number of lines that have not yet been read from **readLine()** (p. 291) by a **Distributor** (p. 303).

Returns

Number of lines that have not been distributed.

G.39.1.4 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. 560)*	Lines not randomized.

G.39.1.5 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.

G.39.1.6 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. 560)*	**Error** (p. 106) with the file stream.
Error::ObjectDoesNotExist *(p. 453)*	File stream or buffer is exhausted.

G.39.1.7 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.

G.39.2 Member Data Documentation

G.39.2.1 CHUNKSIZEPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::CHUNKSIZEPROPERTY [static]`

Number of lines sent in succession

G.39.2.2 DELIMITERPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::DELIMITERPROPERTY [static]`

Delimiter to tokenize sent lines

G.39.2.3 INPUTCSVPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::INPUTCSVPROPERTY [static]`

Text (p. 151) file to read

G.39.2.4 RANDOMIZEPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::RANDOMIZEPROPERTY [static]`

Randomly iterate buffer

G.39.2.5 RANDOMSEEDPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::RANDOMSEEDPROPERTY [static]`

Seed for randomization

G.39.2.6 TRIMPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::TRIMPROPERTY [static]`

Trim whitespace from lines read

G.39.2.7 USEBUFFERPROPERTY

`const std::string BiometricEvaluation::MPI::CSVResources::USEBUFFERPROPERTY [static]`

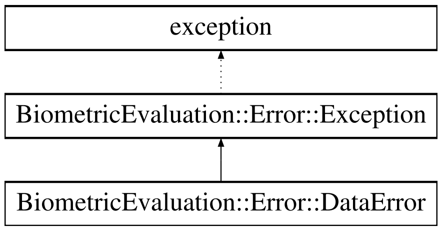
Read file into buffer first, or read from file

G.40 BiometricEvaluation::Error::DataError Class Reference

Error (p. 106) when reading data from an external source.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::DataError:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::DataError

Public Member Functions

- **DataError** ()
- **DataError** (const std::string &info)

G.40.1 Detailed Description

Error (p. 106) 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.

G.40.2 Constructor & Destructor Documentation

G.40.2.1 DataError() `[1/2]`

```
BiometricEvaluation::Error::DataError::DataError ( )
```

Construct a **DataError** (p. 293) object with the default information string.

G.40.2.2 DataError() `[2/2]`

```
BiometricEvaluation::Error::DataError::DataError (
              const std::string & info )
```

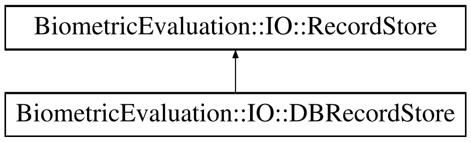
Construct a **DataError** (p. 293) object with an information string appended to the default information string.

G.41 BiometricEvaluation::IO::DBRecordStore Class Reference

A class that implements **IO::RecordStore** (p. 500) using a Berkeley DB database as the underlying record storage system.

`#include <be_io_dbrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::DBRecordStore:


BiometricEvaluation::IO::RecordStore
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. 500), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 500), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  *Move the **RecordStore** (p. 500).*
- 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

Additional Inherited Members

G.41.1 Detailed Description

A class that implements **IO::RecordStore** (p. 500) using a Berkeley DB database as the underlying record storage system.

G.41.2 Constructor & Destructor Documentation

G.41.2.1 DBRecordStore() [1/2]

```
BiometricEvaluation::IO::DBRecordStore::DBRecordStore (
            const std::string & pathname,
            const std::string & description )
```

Create a new **DBRecordStore** (p. 293), read/write mode.

Parameters

in	*pathname*	The directory where the store will be created.
in	*description*	The store's description.

Exceptions

Error::ObjectExists *(p. 454)*	The store already exists.
Error::StrategyError *(p. 560)*	An error occurred when accessing the underlying file system.

G.41.2.2 DBRecordStore() [2/2]

```
BiometricEvaluation::IO::DBRecordStore::DBRecordStore (
             const std::string & pathname,
              IO::Mode mode =  IO::Mode::ReadOnly )
```

Open an existing **DBRecordStore** (p. 293).

Parameters

in	*name*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	The store does not exist.
Error::StrategyError *(p. 560)*	An error occurred when accessing the underlying file system.

G.41.3 Member Function Documentation

G.41.3.1 changeDescription()

```
void BiometricEvaluation::IO::DBRecordStore::changeDescription (
             const std::string & description )  [override], [virtual]
```

Change the description of the **RecordStore** (p. 500).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 502).

G.41.3.2 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.41.3.3 getCount()

```
unsigned int BiometricEvaluation::IO::DBRecordStore::getCount ( ) const  [override], [virtual]
```

Obtain the number of items in the **RecordStore** (p. 500).

Returns

The number of items in the **RecordStore** (p. 500).

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.41.3.4 getDescription()

```
std::string BiometricEvaluation::IO::DBRecordStore::getDescription ( ) const  [override], [virtual]
```

Obtain a textual description of the **RecordStore** (p. 500).

Returns

The **RecordStore** (p. 500)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.41.3.5 getPathname()

```
std::string BiometricEvaluation::IO::DBRecordStore::getPathname ( ) const  [override], [virtual]
```

Return the path name of the **RecordStore** (p. 500).

Returns

Where in the file system the **RecordStore** (p. 500) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.41.3.6 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. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.41.3.7 insert()

```
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. 454)*	A record with the given key is already present.
Error::StrategyError *(p. 560)*	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.41.3.8 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.41.3.9 move()

```
void BiometricEvaluation::IO::DBRecordStore::move (
            const std::string & pathname )  [override], [virtual]
```

Move the **RecordStore** (p. 500).
The **RecordStore** (p. 500) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 507).

G.41.3.10 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 508).

G.41.3.11 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. *453*)	A record for the key does not exist.
Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 509).

G.41.3.12 sequence()

```
RecordStore::Record BiometricEvaluation::IO::DBRecordStore::sequence (
            int cursor =  BE_RECSTORE_SEQ_NEXT )  [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. *453*)	End of sequencing.
Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.41.3.13 sequenceKey()

```
std::string BiometricEvaluation::IO::DBRecordStore::sequenceKey (
            int cursor =  BE_RECSTORE_SEQ_NEXT )  [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.41.3.14 setCursorAtKey()

```
void BiometricEvaluation::IO::DBRecordStore::setCursorAtKey (
            const std::string & key )  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 500), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 299).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 299).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.41.3.15 sync()

```
void BiometricEvaluation::IO::DBRecordStore::sync ( ) const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.42 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**

G.42.1 Detailed Description

Representation of an extended feature set delta.

G.43 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. 302) struct.*

Public Attributes

- **Image::Coordinate coordinate**
- bool **has_angle**
- int **angle1**
- int **angle2**
- int **angle3**

G.43.1 Detailed Description

Representation of the delta.

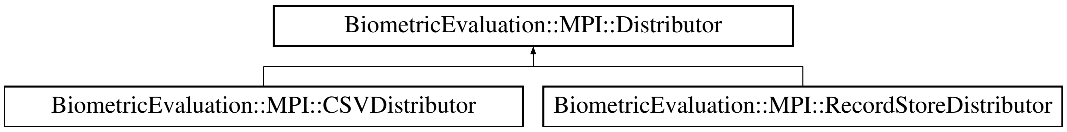
A delta has a coordinate and an optional angle. The units for the X/Y coordinate and the angle are specific to the record format represented by an object of this class.

G.44 BiometricEvaluation::MPI::Distributor Class Reference

A class to represent an **MPI** (p. 143) task that distributes work to other tasks.

`#include <be_mpi_distributor.h>`

Inheritance diagram for BiometricEvaluation::MPI::Distributor:


BiometricEvaluation::MPI::Distributor
BiometricEvaluation::MPI::CSVDistributor
BiometricEvaluation::MPI::RecordStoreDistributor

Public Member Functions

- **Distributor** (const std::string &propertiesFileName)

  Constructor with properties file name.
- void **start** ()

  *Start of **MPI** (p. 143) processing for the distributor.*

Protected Member Functions

- virtual void **createWorkPackage** (**MPI::WorkPackage** &workPackage)=0

  Create a work package for distribution.
- std::shared_ptr< **IO::Logsheet** > **getLogsheet** () const

  Get access to the Logsheet object.

G.44.1 Detailed Description

A class to represent an **MPI** (p. 143) task that distributes work to other tasks.

A **Distributor** (p. 303) object is based on a set of properties contained in a file. This class must be subclassed and an implementation of the **createWorkPackage()** (p. 304) method provided.

The distributor sends an **MPI** (p. 143) 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. 303) object will result in the entire **MPI** (p. 143) 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. 482)
MPI::Receiver (p. 494)
MPI::WorkPackage (p. 600)

G.44.2 Constructor & Destructor Documentation

G.44.2.1 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. 307)*	An error occurred, possibly due to missing or invalid properties.

G.44.3 Member Function Documentation

G.44.3.1 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::RecordStoreDistributor** (p. 514), and **BiometricEvaluation↩ ::MPI::CSVDistributor** (p. 286).

G.44.3.2 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.

G.44.3.3 start()

```
void BiometricEvaluation::MPI::Distributor::start ( )
```

Start of **MPI** (p. 143) processing for the distributor.

Once started, the distributor will send a message to each receiver task telling it to start and waiting for status back from each receiver.

G.45 BiometricEvaluation::DataInterchange::AN2KRecord::Domain↩ Name Struct Reference

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

```
#include <be_data_interchange_an2k.h>
```

Public Member Functions

- **DomainName** (std::string **identifier=""**, std::string **version=""**)

  Create a ***DomainName*** *(p. 305) struct.*

Public Attributes

- std::string **identifier**
- std::string **version**

G.45.1 Detailed Description

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

G.45.2 Constructor & Destructor Documentation

G.45.2.1 DomainName()

```
BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::DomainName (
              std::string identifier = "",
              std::string version = "" )  [inline]
```

Create a **DomainName** (p. 305) struct.

Parameters

identifier	Unique identifier for agency, entity, or implementation.
version	Optional unique version number of the implementation of the identifier.

G.45.3 Member Data Documentation

G.45.3.1 identifier

```
std::string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::identifier
```

Unique identifier for agency, entity, or implementation.

G.45.3.2 version

```
std::string BiometricEvaluation::DataInterchange::AN2KRecord::DomainName::version
```

Optional version of the implementation

G.46 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification↩ ::Entry Struct Reference

Public Member Functions

- **Entry** (bool **standard**, std::string **code**)

Public Attributes

- bool **standard**
- std::string **code**

G.46.1 Constructor & Destructor Documentation

G.46.1.1 Entry()

```
BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::Entry (
            bool standard,
            std::string code )
```

Create an **Entry** (p. 306) struct.

Parameters

standard	Whether or not code is a standard AN2K pattern classification code.
code	AN2K or user-defined pattern classification code.

G.46.2 Member Data Documentation

G.46.2.1 code

```
std::string BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::code
```

AN2K or user-defined pattern classification code.

G.46.2.2 standard

```
bool BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification::Entry::standard
```

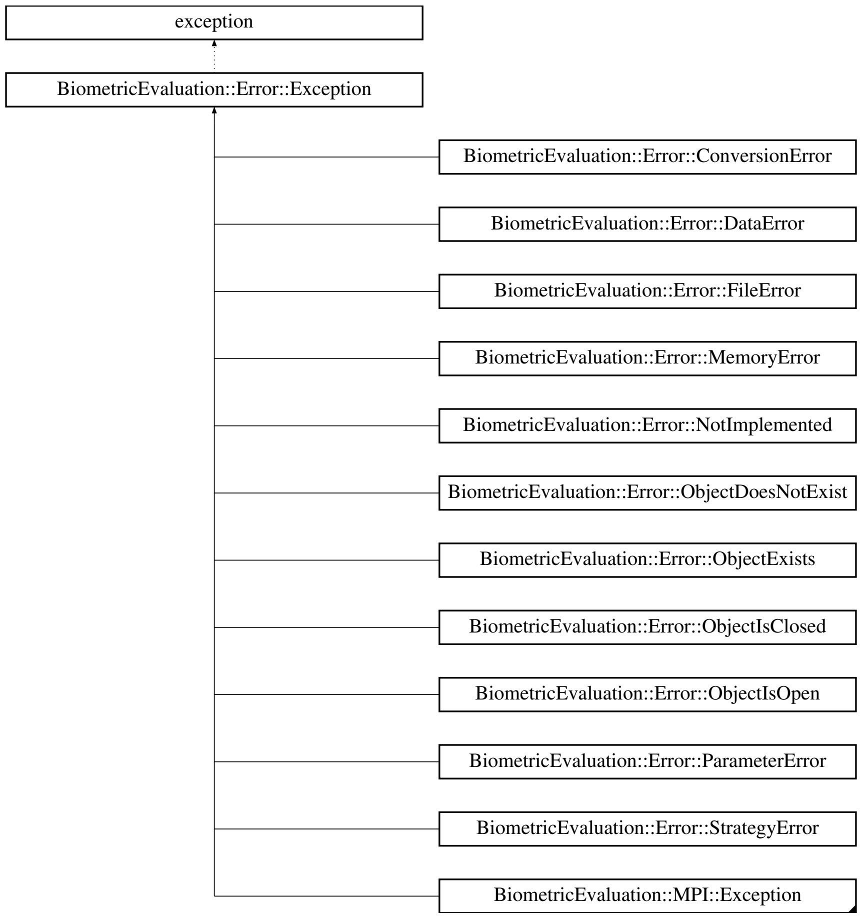
Whether code is a standard AN2K pattern classification code.

G.47 BiometricEvaluation::Error::Exception Class Reference

The parent class of all **BiometricEvaluation** (p. 105) exceptions.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::Exception:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ConversionError
BiometricEvaluation::Error::DataError
BiometricEvaluation::Error::FileError
BiometricEvaluation::Error::MemoryError
BiometricEvaluation::Error::NotImplemented
BiometricEvaluation::Error::ObjectDoesNotExist
BiometricEvaluation::Error::ObjectExists
BiometricEvaluation::Error::ObjectIsClosed
BiometricEvaluation::Error::ObjectIsOpen
BiometricEvaluation::Error::ParameterError
BiometricEvaluation::Error::StrategyError
BiometricEvaluation::MPI::Exception

Public Member Functions

- **Exception** ()
- **Exception** (std::string info)
- const char ∗ **what** () const noexcept
- const std::string **whatString** () const noexcept

G.47.1 Detailed Description

The parent class of all **BiometricEvaluation** (p. 105) 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.

G.47.2 Constructor & Destructor Documentation

G.47.2.1 Exception() `[1/2]`

```
BiometricEvaluation::Error::Exception::Exception ( )
```

Construct an **Exception** (p. 307) object without an information string.

G.47.2.2 Exception() `[2/2]`

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

Construct an **Exception** (p. 307) object with an information string.

Parameters

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

G.47.3 Member Function Documentation

G.47.3.1 what()

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

Obtain the information string associated with the exception.

Returns

The information string as a char array.

G.47.3.2 whatString()

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

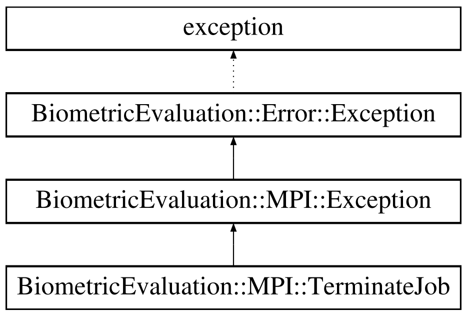
Obtain the information string associated with the exception.

Returns

The information string.

G.48 BiometricEvaluation::MPI::Exception Class Reference

Inheritance diagram for BiometricEvaluation::MPI::Exception:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::MPI::Exception
BiometricEvaluation::MPI::TerminateJob

Public Member Functions

- **Exception** ()
- **Exception** (std::string info)

  Constructor.
- virtual ∼**Exception** () noexcept=default

G.48.1 Constructor & Destructor Documentation

G.48.1.1 Exception() [1/2]

```
BiometricEvaluation::MPI::Exception::Exception ( )
```

Construct with default information string.

G.48.1.2 Exception() [2/2]

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

Constructor.

Parameters

info	Custom information string. Will be appended to the default information string.

G.48.1.3 ∼Exception()

```
virtual BiometricEvaluation::MPI::Exception::∼Exception ( )  [virtual], [default], [noexcept]
```

Destructor.

Reimplemented from **BiometricEvaluation::Error::Exception** (p. 307).

G.49 BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeatureSet Class Reference

A class to represent the Extended **Feature** (p. 109) 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** ()

  *Obtain the structure containing information about the image and Extended **Feature** (p. 109) Set.*
- BiometricEvaluation::Feature::AN2K11EFS::MinutiaPointSet **getMPS** ()

  Obtain the minutiae point set.

- **BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCountInfo getMRCI** ()

  Obtain all the information relating to minutiae ridge count information.
- BiometricEvaluation::Feature::AN2K11EFS::CorePointSet **getCPS** ()

  Obtain the core point set.
- BiometricEvaluation::Feature::AN2K11EFS::DeltaPointSet **getDPS** ()

  Obtain the delta point set.
- **BiometricEvaluation::Feature::AN2K11EFS::NoFeaturesPresent getNFP** ()

G.49.1 Detailed Description

A class to represent the Extended **Feature** (p. 109) 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.

G.49.2 Constructor & Destructor Documentation

G.49.2.1 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.
in	*recordNumber*	Which fingerprint minutiae record to read from the complete AN2K record.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	The named file does not exist.
Error::StrategyError *(p. 560)*	An error occurred when opening or reading from the file.
Error::DataError *(p. 293)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.

G.49.2.2 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.
in	*recordNumber*	Which fingerprint minutiae record to read from the complete AN2K record.

Exceptions

Error::DataError *(p. 293)*	An error occurred reading the AN2K record, or there is no fingerprint minutiae record for the requested number.

G.49.3 Member Function Documentation

G.49.3.1 getCPS()

```
BiometricEvaluation::Feature::AN2K11EFS::CorePointSet BiometricEvaluation::Feature::AN2K11↩
EFS::ExtendedFeatureSet::getCPS ( )
```

Obtain the core point set.

The set may be empty as this Type-9 field is optional.

Returns

The set of core points.

G.49.3.2 getDPS()

```
BiometricEvaluation::Feature::AN2K11EFS::DeltaPointSet BiometricEvaluation::Feature::AN2K11↩
EFS::ExtendedFeatureSet::getDPS ( )
```

Obtain the delta point set.

The set may be empty as this Type-9 field is optional.

Returns

The set of delta points.

G.49.3.3 getImageInfo()

```
ImageInfo BiometricEvaluation::Feature::AN2K11EFS::ExtendedFeatureSet::getImageInfo ( )
```

Obtain the structure containing information about the image and Extended **Feature** (p. 109) Set.

Returns

The information about the image.

G.49.3.4 getMPS()

`BiometricEvaluation::Feature::AN2K11EFS::MinutiaPointSet BiometricEvaluation::Feature::AN2↩ K11EFS::ExtendedFeatureSet::getMPS ( )`

Obtain the minutiae point set.

The set may be empty as this Type-9 field is optional.

Returns

The set of minutia points.

G.49.3.5 getMRCI()

`**BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCountInfo** BiometricEvaluation::Feature↩ ::AN2K11EFS::ExtendedFeatureSet::getMRCI ( )`

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.

G.49.3.6 getNFP()

`**BiometricEvaluation::Feature::AN2K11EFS::NoFeaturesPresent** BiometricEvaluation::Feature::↩ AN2K11EFS::ExtendedFeatureSet::getNFP ( )`

Obtain the No Features Present indicators.

Returns

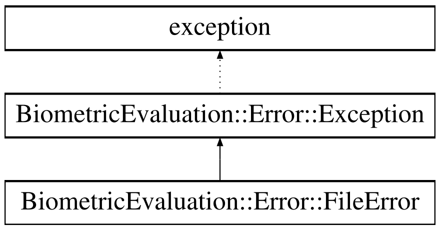
The flags for No Features Present.

G.50 BiometricEvaluation::Error::FileError Class Reference

File error when opening, reading, writing, etc.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::FileError:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::FileError

Public Member Functions

- **FileError** ()
- **FileError** (const std::string &info)

G.50.1 Detailed Description

File error when opening, reading, writing, etc.

G.50.2 Constructor & Destructor Documentation

G.50.2.1 FileError() [1/2]

```
BiometricEvaluation::Error::FileError::FileError ( )
```

Construct a **FileError** (p. 312) object with the default information string.

G.50.2.2 FileError() [2/2]

```
BiometricEvaluation::Error::FileError::FileError (
            const std::string & info )
```

Construct a **FileError** (p. 312) object with an information string appended to the default information string.

G.51 BiometricEvaluation::IO::FileLogCabinet Class Reference

```
#include <be_io_filelogcabinet.h>
```

Public Member Functions

- **FileLogCabinet** (const std::string &pathname, const std::string &description)
- **FileLogCabinet** (const std::string &pathname)
- std::shared_ptr< **FileLogsheet** > **newLogsheet** (const std::string &name, const std::string &description)
- std::string **getPathname** ()
- std::string **getDescription** ()
- unsigned int **getCount** ()

G.51.1 Detailed Description

A class to represent a collection of log sheets.

G.51.2 Constructor & Destructor Documentation

G.51.2.1 FileLogCabinet() [1/2]

```
BiometricEvaluation::IO::FileLogCabinet::FileLogCabinet (
            const std::string & pathname,
            const std::string & description )
```

Create a new **FileLogCabinet** (p. 313) in the file system.

Parameters

in	*pathname*	The pathname where the **FileLogCabinet** (p. 313) is to be created.
in	*description*	The text used to describe the cabinet.

Exceptions

Error::ObjectExists (p. *454*)	The cabinet was previously created.
Error::StrategyError (p. *560*)	An error occurred when using the underlying file system.

G.51.2.2 FileLogCabinet() [2/2]

```
BiometricEvaluation::IO::FileLogCabinet::FileLogCabinet (
             const std::string & pathname )
```

Open an existing **FileLogCabinet** (p. 313).

Parameters

in	*pathname*	The pathname where the **FileLogCabinet** (p. 313) is located.

Exceptions

Error::ObjectDoesNotExist (p. *453*)	The cabinet does not exist in the file system.
Error::StrategyError (p. *560*)	An error occurred when using the underlying file system.

G.51.3 Member Function Documentation

G.51.3.1 getCount()

```
unsigned int BiometricEvaluation::IO::FileLogCabinet::getCount ( )
```

Obtain the number of items in the **FileLogCabinet** (p. 313).
@ returns The number of logsheets manages by the cabinet.

G.51.3.2 getDescription()

```
std::string BiometricEvaluation::IO::FileLogCabinet::getDescription ( )
```

Obtain the description of the **FileLogCabinet** (p. 313).
@ returns The description of the **FileLogCabinet** (p. 313).

G.51.3.3 getPathname()

```
std::string BiometricEvaluation::IO::FileLogCabinet::getPathname ( )
```

Obtain the pathname of the **FileLogCabinet** (p. 313).
@ returns The pathname of the **FileLogCabinet** (p. 313).

G.51.3.4 newLogsheet()

```
std::shared_ptr< FileLogsheet> BiometricEvaluation::IO::FileLogCabinet::newLogsheet (
             const std::string & name,
             const std::string & description )
```

Create a new **FileLogsheet** (p. 315) within the cabinet.

Parameters

in	*name*	The name of the **FileLogsheet** (p. 315) to be created. This can not be a path name.
in	*description*	The text used to describe the sheet. This text is written into the log file prior to any entries.

Returns

An object pointer to the new log sheet.

Exceptions

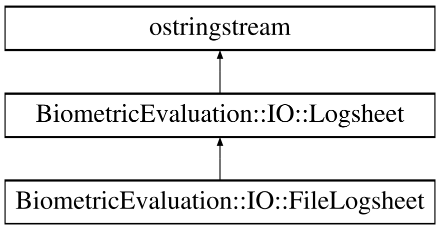
Error::ObjectExists *(p. 454)*	The sheet was previously created.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying file system.

G.52 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:


ostringstream
BiometricEvaluation::IO::Logsheet
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. 315), returning one entry per invocation.*
- void **write** (const std::string &entry)

  Write a string as an entry to the backing store.
- void **writeComment** (const std::string &entry)

  Write a string as a comment to the backing store.

- void **writeDebug** (const std::string &entry)

  Write a string as a debug entry to the backing store.
- void **sync** ()

  Synchronize any buffered data to the underlying backing store.

Static Public Member Functions

- static void **mergeLogsheets** (std::vector< std::shared_ptr< **FileLogsheet** >> &logsheets)

  *Merge multiple FileLogsheets into a single **FileLogsheet** (p. 315).*
- static std::string **trim** (const std::string &entry)

  *Trim delimiters from **FileLogsheet** (p. 315) entries.*

Static Public Attributes

- static const int32_t **BE_FILELOGSHEET_SEQ_START** = 1
- static const int32_t **BE_FILELOGSHEET_SEQ_NEXT** = 2

Protected Member Functions

- **FileLogsheet** (const **FileLogsheet** &)
- **FileLogsheet** & **operator=** (const **FileLogsheet** &)
- void **updateCursor** ()

  Update the cursor position of the sequence file.

Protected Attributes

- std::unique_ptr< std::fstream > **_theLogFile**
- std::shared_ptr< std::fstream > **_sequenceFile**
- streamoff **_cursor**

Additional Inherited Members

G.52.1 Detailed Description

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

A **FileLogsheet** (p. 315) 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.

G.52.2 Constructor & Destructor Documentation

G.52.2.1 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 **FileLogsheet** (p. 315) to be created.
`in`	*description*	The text used to describe the sheet. This text is written into the log file prior to any entries.

Exceptions

Error::ParameterError *(p. 470)*	The URL is malformed.
Error::ObjectExists *(p. 454)*	The sheet was previously created.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying file system, or name or parentDir is malformed.

G.52.2.2 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. 315) may be a costly operation.

Parameters

`in`	*url*	The Uniform Resource Locator of the **FileLogsheet** (p. 315) to be opened.

Exceptions

Error::ParameterError *(p. 470)*	The URL is malformed.
Error::ObjectDoesNotExist *(p. 453)*	The sheet does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying file system, or name or parentDir is malformed.

G.52.2.3 ∼FileLogsheet()

```
BiometricEvaluation::IO::FileLogsheet::~FileLogsheet ( )
```

Destructor

G.52.2.4 FileLogsheet() `[3/3]`

```
BiometricEvaluation::IO::FileLogsheet::FileLogsheet (
            const FileLogsheet & ) [protected]
```

Prevent copying of **FileLogsheet** (p. 315) objects

G.52.3 Member Function Documentation

G.52.3.1 mergeLogsheets()

```
static void BiometricEvaluation::IO::FileLogsheet::mergeLogsheets (
            std::vector< std::shared_ptr< FileLogsheet >> & logsheets )  [static]
```

Merge multiple FileLogsheets into a single **FileLogsheet** (p. 315).
Logsheet (p. 416) 2 - n will be appended to **Logsheet** (p. 416) 1.

Parameters

logSheets	**Logsheet** (p. 416) to merge.

Exceptions

Error::FileError *(p. 312)*	**Error** (p. 106) during log sequence.
Error::StrategyError *(p. 560)*	**Error** (p. 106) during log sequence.

G.52.3.2 operator=()

```
FileLogsheet& BiometricEvaluation::IO::FileLogsheet::operator= (
            const FileLogsheet & )  [protected]
```

Prevent copying of **FileLogsheet** (p. 315) objects

G.52.3.3 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. 315), 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. 319).

Exceptions

Error::FileError *(p. 312)*,***Error*** *(p. 106)*	occured while performing file **IO** (p. 124).
Error::ObjectDoesNotExist *(p. 453)*	The **FileLogsheet** (p. 315) cannot be found on disk.
Error::StrategyError *(p. 560)*	Invalid cursor position or the contents of the **FileLogsheet** (p. 315) is malformed.

G.52.3.4 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. 560)*	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 423).

G.52.3.5 trim()

```
static std::string BiometricEvaluation::IO::FileLogsheet::trim (
             const std::string & entry )  [static]
```

Trim delimiters from **FileLogsheet** (p. 315) entries.
Works for comments and numbered entries.

Parameters

in	*entry*	The entry to trim.

Returns

Delimiter-less entry.

G.52.3.6 updateCursor()

```
void BiometricEvaluation::IO::FileLogsheet::updateCursor ( )  [protected]
```

Update the cursor position of the sequence file.

Exceptions

Error::FileError *(p. 312)*	**Error** (p. 106) getting file position from sequence file.

G.52.3.7 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. 560)	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 423).

G.52.3.8 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. 560)	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 424).

G.52.3.9 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. 560)*	An error occurred when logging.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 424).

G.52.4 Member Data Documentation

G.52.4.1 _cursor

`streamoff BiometricEvaluation::IO::FileLogsheet::_cursor [protected]`

Position of the sequencer, relative to SOF

G.52.4.2 _sequenceFile

`std::shared_ptr<std::fstream> BiometricEvaluation::IO::FileLogsheet::_sequenceFile [protected]`

Stream used for sequencing

G.52.4.3 _theLogFile

`std::unique_ptr<std::fstream> BiometricEvaluation::IO::FileLogsheet::_theLogFile [protected]`

Stream used for writing the log file

G.52.4.4 BE_FILELOGSHEET_SEQ_NEXT

`const int32_t BiometricEvaluation::IO::FileLogsheet::BE_FILELOGSHEET_SEQ_NEXT = 2 [static]`

Sequence from current position

G.52.4.5 BE_FILELOGSHEET_SEQ_START

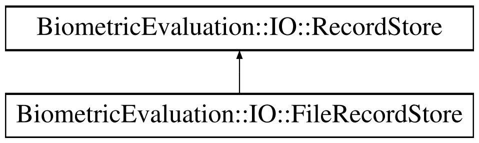
`const int32_t BiometricEvaluation::IO::FileLogsheet::BE_FILELOGSHEET_SEQ_START = 1 [static]`

Sequence from beginning

G.53 BiometricEvaluation::IO::FileRecordStore Class Reference

`#include <be_io_filerecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::FileRecordStore:


BiometricEvaluation::IO::RecordStore
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. 500), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 500), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  *Move the **RecordStore** (p. 500).*
- 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

Additional Inherited Members

G.53.1 Detailed Description

Class to represent the record store data storage mechanism implemented as files for each record.

Note

For the methods that take a key parameter, **Error::StrategyError** (p. 560) will be thrown if the key string is not compliant. A **FileRecordStore** (p. 321) has the additional requirement that a key name may not contain path delimiter characters ('/' and '\'), or begin with whitespace.

G.53.2 Constructor & Destructor Documentation

G.53.2.1 FileRecordStore() [1/2]

```
BiometricEvaluation::IO::FileRecordStore::FileRecordStore (
            const std::string & pathname,
            const std::string & description )
```

Create a new **FileRecordStore** (p. 321), 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. 454)*	The store already exists.
Error::StrategyError *(p. 560)*	An error occurred when accessing the underlying file system.

G.53.2.2 FileRecordStore() [2/2]

```
BiometricEvaluation::IO::FileRecordStore::FileRecordStore (
             const std::string & name,
              IO::Mode mode =  IO::Mode::ReadOnly )
```

Open an existing **FileRecordStore** (p. 321).

Parameters

in	*name*	The path name of the store.
in	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	The store does not exist.
Error::StrategyError *(p. 560)*	An error occurred when accessing the underlying file system.

G.53.3 Member Function Documentation

G.53.3.1 changeDescription()

```
void BiometricEvaluation::IO::FileRecordStore::changeDescription (
             const std::string & description )  [override], [virtual]
```

Change the description of the **RecordStore** (p. 500).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 502).

G.53.3.2 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.53.3.3 getCount()

```
unsigned int BiometricEvaluation::IO::FileRecordStore::getCount ( ) const  [override], [virtual]
```

Obtain the number of items in the **RecordStore** (p. 500).

Returns

The number of items in the **RecordStore** (p. 500).

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.53.3.4 getDescription()

```
std::string BiometricEvaluation::IO::FileRecordStore::getDescription ( ) const  [override], [virtual]
```

Obtain a textual description of the **RecordStore** (p. 500).

Returns

The **RecordStore** (p. 500)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.53.3.5 getPathname()

```
std::string BiometricEvaluation::IO::FileRecordStore::getPathname ( ) const  [override], [virtual]
```

Return the path name of the **RecordStore** (p. 500).

Returns

Where in the file system the **RecordStore** (p. 500) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.53.3.6 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. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.53.3.7 insert()

```
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. 454)*	A record with the given key is already present.
Error::StrategyError *(p. 560)*	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.53.3.8 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.53.3.9 move()

```
void BiometricEvaluation::IO::FileRecordStore::move (
             const std::string & pathname )  [override], [virtual]
```

Move the **RecordStore** (p. 500).
The **RecordStore** (p. 500) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 507).

G.53.3.10 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 508).

G.53.3.11 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. *453*)	A record for the key does not exist.
Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 509).

G.53.3.12 replace()

```
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. 500).

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. *453*)	A record for the key does not exist.
Error::StrategyError (p. *560*)	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 510).

G.53.3.13 sequence()

```
RecordStore::Record BiometricEvaluation::IO::FileRecordStore::sequence (
             int cursor = BE_RECSTORE_SEQ_NEXT )  [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.53.3.14 sequenceKey()

```
std::string BiometricEvaluation::IO::FileRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT ) [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.53.3.15 setCursorAtKey()

```
void BiometricEvaluation::IO::FileRecordStore::setCursorAtKey (
            const std::string & key )  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 500), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 327).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 327).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.53.3.16 sync()

```
void BiometricEvaluation::IO::FileRecordStore::sync ( ) const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.54 BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReading↩ System 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**

G.54.1 Detailed Description

Representation of information about a fingerprint reader system.

G.54.2 Member Data Documentation

G.54.2.1 equipment

`std::string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::equipment`

Optional ID for equipment used in system

G.54.2.2 method

`EncodingMethod BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::method`

Method used to encoded minutiae

G.54.2.3 name

`std::string BiometricEvaluation::Feature::AN2K7Minutiae::FingerprintReadingSystem::name`

Name for system that encoded minutiae

G.55 BiometricEvaluation::Finger::AN2KViewCapture::FingerSegment↩ Position 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. 330) struct.*

Public Attributes

- **Finger::Position fingerPosition**
- Image::CoordinateSet **coordinates**

G.55.1 Detailed Description

Locations of an individual finger segment in a slap.

G.55.2 Constructor & Destructor Documentation

G.55.2.1 FingerSegmentPosition()

```
BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::FingerSegmentPosition (
            const Finger::Position fingerPosition,
            const Image::CoordinateSet coordinates )
```

Create an **FingerSegmentPosition** (p. 330) struct.

Parameters

fingerPosition	**Finger** (p. 113) depicted in this segment.
coordinates	Collection of coordinates that compose the segment bonding polygon.

G.55.3 Member Data Documentation

G.55.3.1 coordinates

`Image::CoordinateSet BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::coordinates`

Points composing the segmented polygon

G.55.3.2 fingerPosition

`Finger::Position BiometricEvaluation::Finger::AN2KViewCapture::FingerSegmentPosition::finger↩ Position`

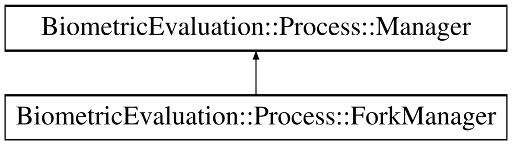
Finger (p. 113) depicted in this segment

G.56 BiometricEvaluation::Process::ForkManager Class Reference

Manager (p. 426) implementation that starts Workers by calling fork(2).

`#include <be_process_forkmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::ForkManager:


BiometricEvaluation::Process::Manager
BiometricEvaluation::Process::ForkManager

Public Member Functions

- **ForkManager** ()
- std::shared_ptr< **WorkerController** > **addWorker** (std::shared_ptr< **Worker** > worker)

  *Adds a **Worker** (p. 588) to be managed by this **Manager** (p. 426).*
- void **startWorkers** (bool wait=true, bool communicate=false)

  *Begin **Worker** (p. 588)'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. 588) 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. 331) is responsbile for a particular PID.*
- void **setNotWorking** (const pid_t pid)

  Set Status.isWorking for PID to false.
- void **markAllFinished** ()

  *Call **setNotWorking()** (p. 335) for all PIDs known to this **ForkManager** (p. 331).*
- bool **getIsWorkingStatus** (const pid_t pid) const

Get Status.isWorking for PID.
- void **waitForWorkerExit** ()
  Block until all Workers have exited.
- ∼**ForkManager** ()
  ***ForkManager** (p. 331) 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. 595) for given process ID.*

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

G.56.1 Detailed Description

Manager (p. 426) implementation that starts Workers by calling fork(2).

G.56.2 Constructor & Destructor Documentation

G.56.2.1 ForkManager()

```
BiometricEvaluation::Process::ForkManager::ForkManager ( )
```

ForkManager (p. 331) constructor.

G.56.3 Member Function Documentation

G.56.3.1 addWorker()

```
std::shared_ptr< WorkerController> BiometricEvaluation::Process::ForkManager::addWorker (
            std::shared_ptr< Worker > worker ) [virtual]
```

Adds a **Worker** (p. 588) to be managed by this **Manager** (p. 426).

Parameters

worker	A **Worker** (p. 588) instance to run.

Returns

shared_ptr to worker.

Implements **BiometricEvaluation::Process::Manager** (p. 427).

G.56.3.2 broadcastSignal()

```
void BiometricEvaluation::Process::ForkManager::broadcastSignal (
             int signo )
```

Send a POSIX signal to all workers.

Parameters

in	*signo*	The signal to send.

G.56.3.3 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. 337) object that exited.
status	The status of the **Worker** (p. 588) that exited (from wait(2)).

G.56.3.4 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. 453)*	PID not under this manager's control.

G.56.3.5 responsibleFor()

```
bool BiometricEvaluation::Process::ForkManager::responsibleFor (
             const pid_t pid ) const
```

Obtain whether or not this **ForkManager** (p. 331) is responsbile for a particular PID.

Parameters

in	*pid*	PID in question

Returns

true if this **ForkManager** (p. 331) spawned pid, false otherwise.

G.56.3.6 setExitCallback()

```
void BiometricEvaluation::Process::ForkManager::setExitCallback (
             void(*)(std::shared_ptr< ForkWorkerController > worker, int stat_loc) exitCallback
)
```

Call a function in your program when a child exits.

Parameters

exitCallback	Function pointer to a method that takes a shared_ptr to a **ForkWorkerController** (p. 337) and the integer status information.

Note

The exit callback will not have any effect if the **Manager** (p. 426) is not set to wait for Workers.

G.56.3.7 setExitStatus()

```
void BiometricEvaluation::Process::ForkManager::setExitStatus (
             const pid_t pid,
             const int32_t waitStatus )
```

Set the exit status in the **WorkerController** (p. 595) 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. 453)*	PID not under this manager's control.

Note

Exit status is only set if process exited cleanly.

G.56.3.8 setNotWorking()

```
void BiometricEvaluation::Process::ForkManager::setNotWorking (
             const pid_t pid )
```

Set Status.isWorking for PID to false.

Parameters

in	*pid*	PID whose inWorking flag should be set to false

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	PID not under this manager's control.

G.56.3.9 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. 595) that is being managed by this **Manager** (p. 426) instance.
	wait	Whether or not to wait for this **Worker** (p. 588) 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. 454)*	worker is already working.
Error::StrategyError *(p. 560)*	worker is not managed by this **Manager** (p. 426) instance.

Implements **BiometricEvaluation::Process::Manager** (p. 429).

G.56.3.10 startWorkers()

```
void BiometricEvaluation::Process::ForkManager::startWorkers (
            bool wait = true,
            bool communicate = false )  [virtual]
```

Begin **Worker** (p. 588)'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. 454)*	At least one **Worker** (p. 588) is already working.
Error::StrategyError *(p. 560)*	Problem forking.

Implements **BiometricEvaluation::Process::Manager** (p. 430).

G.56.3.11 stopWorker()

```
void BiometricEvaluation::Process::ForkManager::stopWorker (
            std::shared_ptr< WorkerController > workerController )  [virtual]
```

Ask **Worker** (p. 588) to exit.

Sends SIGUSR1 to the **Worker** (p. 588), which **ForkManager** (p. 331) will handle automatically.

Parameters

workerController	Pointer to the **ForkWorkerController** (p. 337) that should be stopped.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	worker is not working.
Error::StrategyError *(p. 560)*	Problem sending the signal.

Attention

Do not call **stopWorker()** (p. 336) 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. 588) exits.

Implements **BiometricEvaluation::Process::Manager** (p. 430).

G.56.3.12 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. 431).

G.56.4 Member Data Documentation

G.56.4.1 FORKMANAGERS

`std::list< ForkManager*> BiometricEvaluation::Process::ForkManager::FORKMANAGERS [static]`

List of all instantiated ForkManagers.

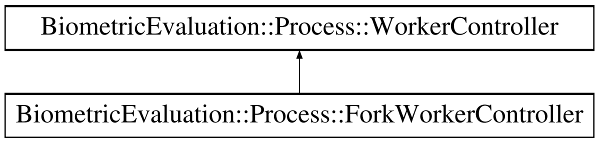
This is not a list of managed pointers to ForkManagers. If it was, the smart pointer's destructor would attempt to delete the object being pointed to at program termination, which is ultimately sometime after the destructor of the **ForkManager** (p. 331) itself was called.

G.57 BiometricEvaluation::Process::ForkWorkerController Class Reference

Wrapper of a **Worker** (p. 588) returned from a **Process::ForkManager** (p. 331).

`#include <be_process_forkmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::ForkWorkerController:


BiometricEvaluation::Process::WorkerController
BiometricEvaluation::Process::ForkWorkerController

Public Member Functions

- bool **isWorking** () const

  *Obtain whether or not **Worker** (p. 588) is working.*
- bool **everWorked** () const

  *Obtain whether or not this **Worker** (p. 588) has ever worked.*
- void **reset** ()

  *Reuse the **Worker** (p. 588).*
- pid_t **getPID** () const

  Obtain the PID of this process this instance represents.
- ~**ForkWorkerController** ()

  ***ForkWorkerController** (p. 337) 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. 588)'s work.*
- void **ForkManager::startWorker** (std::shared_ptr< **WorkerController** > worker, bool wait, bool communicate)

*Restart a completed **Worker** (p. 588).*

- void **ForkManager::stopWorker** (std::shared_ptr< **WorkerController** > workerController)

  *Ask **Worker** (p. 588) to exit.*
- std::shared_ptr< **WorkerController** > **ForkManager::addWorker** (std::shared_ptr< **Worker** > worker)

  *Adds a **Worker** (p. 588) to be managed by this **Manager** (p. 426).*
- void **ForkManager::setExitStatus** (const pid_t pid, const int32_t waitStatus)

  *Set the exit status in the **WorkerController** (p. 595) for given process ID.*

Additional Inherited Members

G.57.1 Detailed Description

Wrapper of a **Worker** (p. 588) returned from a **Process::ForkManager** (p. 331).

G.57.2 Member Function Documentation

G.57.2.1 _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).

G.57.2.2 everWorked()

```
bool BiometricEvaluation::Process::ForkWorkerController::everWorked ( ) const  [virtual]
```

Obtain whether or not this **Worker** (p. 588) has ever worked.

Returns

true the **Worker** (p. 588) has ever or is currently working, false otherwise.

Note

reset() (p. 339) will change the result of this method.

Implements **BiometricEvaluation::Process::WorkerController** (p. 596).

G.57.2.3 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.

G.57.2.4 isWorking()

```
bool BiometricEvaluation::Process::ForkWorkerController::isWorking ( ) const  [virtual]
```

Obtain whether or not **Worker** (p. 588) is working.

Returns

Whether or not the **Worker** (p. 588) is working.

Implements **BiometricEvaluation::Process::WorkerController** (p. 597).

G.57.2.5 reset()

```
void BiometricEvaluation::Process::ForkWorkerController::reset ( )  [virtual]
```

Reuse the **Worker** (p. 588).

Exceptions

Error::ObjectExists *(p. 454)*	The previously started **Worker** (p. 588) is still running.

Reimplemented from **BiometricEvaluation::Process::WorkerController** (p. 597).

G.57.3 Friends And Related Function Documentation

G.57.3.1 ForkManager::addWorker

```
std::shared_ptr< WorkerController> ForkManager::addWorker (
            std::shared_ptr< Worker > worker )  [friend]
```

Adds a **Worker** (p. 588) to be managed by this **Manager** (p. 426).

Parameters

worker	A **Worker** (p. 588) instance to run.

Returns

shared_ptr to worker.

G.57.3.2 ForkManager::setExitStatus

```
void  ForkManager::setExitStatus (
            const pid_t pid,
            const int32_t waitStatus )  [friend]
```

Set the exit status in the **WorkerController** (p. 595) 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. 453)*	PID not under this manager's control.

Note

Exit status is only set if process exited cleanly.

G.57.3.3 ForkManager::startWorker

```
void  ForkManager::startWorker (
            std::shared_ptr< WorkerController > worker,
            bool wait,
            bool communicate )  [friend]
```

Restart a completed **Worker** (p. 588).

Parameters

	worker	Pointer to a **WorkerController** (p. 595) that is being managed by this **Manager** (p. 426) instance.
	wait	Whether or not to wait for this **Worker** (p. 588) 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. 454)*	worker is already working.
Error::StrategyError *(p. 560)*	worker is not managed by this **Manager** (p. 426) instance.

G.57.3.4 ForkManager::startWorkers

```
void ForkManager::startWorkers (
            bool wait,
            bool communicate )  [friend]
```

Begin **Worker** (p. 588)'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. 454)*	One or more of the Workers is already working.
Error::StrategyError *(p. 560)*	Problem forking.

G.57.3.5 ForkManager::stopWorker

```
void ForkManager::stopWorker (
            std::shared_ptr< WorkerController > workerController )  [friend]
```

Ask **Worker** (p. 588) to exit.

Sends SIGUSR1 to the **Worker** (p. 588), which **ForkManager** (p. 331) will handle automatically.

Parameters

workerController	Pointer to the **ForkWorkerController** (p. 337) that should be stopped.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	worker is not working.
Error::StrategyError *(p. 560)*	Problem sending the signal.

G.58 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**

G.58.1 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.

G.58.2 Member Data Documentation

G.58.2.1 fgp

```
Feature::FGP BiometricEvaluation::Feature::AN2K11EFS::FPPPosition::fgp
```

The friction ridge generalized position

G.58.2.2 fsm

```
FingerprintSegment BiometricEvaluation::Feature::AN2K11EFS::FPPPosition::fsm
```

The finger segment position

G.58.2.3 ocf

```
OffCenterFingerPosition BiometricEvaluation::Feature::AN2K11EFS::FPPPosition::ocf
```

The off-center fingerprint position

G.58.2.4 sgp

```
BiometricEvaluation::Image::CoordinateSet BiometricEvaluation::Feature::AN2K11EFS::FPPPosition↩
::sgp
```

The segment polygon

G.59 BiometricEvaluation::Video::Frame Struct Reference

Public Attributes

- **Image::Size size**
- int64_t **timestamp**
- **Memory::uint8Array data**

G.60 BiometricEvaluation::Feature::FrictionRidgeGeneralizedPosition 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**

G.60.1 Detailed Description

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

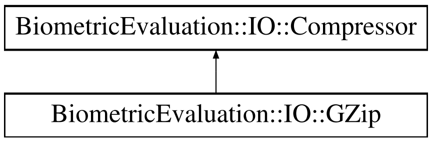
When the AN2K11 FGP field is read, it may represent a finger, palm, or plantar position. The union is tagged to indicate which position is present.

G.61 BiometricEvaluation::IO::GZip Class Reference

Compressor (p. 269) for gzip compression from zlib.

`#include <be_io_gzip.h>`

Inheritance diagram for BiometricEvaluation::IO::GZip:


BiometricEvaluation::IO::Compressor
BiometricEvaluation::IO::GZip

Public Member Functions

- **Memory::uint8Array compress** (const uint8_t ∗const uncompressedData, uint64_t uncompressed←DataSize) 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 compressedData←Size) const

  Decompress a compressed buffer.

- **Memory::uint8Array decompress** (const **Memory::uint8Array** &compressedData) const

  Decompress a compressed buffer.
- **Memory::uint8Array decompress** (const std::string &input) const

  Decompress a compressed buffer into a file.
- void **decompress** (const std::string &inputFile, const std::string &outputFile) const

  Decompress a file.
- void **decompress** (const uint8_t ∗const compressedData, const uint64_t compressedDataSize, const std↩ ::string &outputFile) const

  Decompress a file.
- void **decompress** (const **Memory::uint8Array** &compressedData, const std::string &outputFile) const

  Decompress a file.
- **GZip** (const **GZip** &other)=delete

  Copy constructor (disabled).
- **GZip** & **operator=** (const **GZip** &other)=delete

  Assignment overload (disabled).

Static Public Attributes

- static const std::string **COMPRESSION_LEVEL**
- static const std::string **COMPRESSION_STRATEGY**
- static const std::string **COMPRESSION_METHOD**
- static const std::string **INPUT_DATA_TYPE**
- static const std::string **WINDOW_BITS**
- static const std::string **MEMORY_LEVEL**
- static const std::string **CHUNK_SIZE**

Additional Inherited Members

G.61.1 Detailed Description

Compressor (p. 269) for gzip compression from zlib.

G.61.2 Constructor & Destructor Documentation

G.61.2.1 GZip()

```
BiometricEvaluation::IO::GZip::GZip (
             const  GZip & other )  [delete]
```

Copy constructor (disabled).
Disabled because **Properties** (p. 482) member of parent cannot be copied.

Parameters

other	**GZip** (p. 343) to copy.

G.61.3 Member Function Documentation

G.61.3.1 compress() [1/6]

```
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 uncompressedData.

Returns

Compressed buffer.

Exceptions

Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 271).

G.61.3.2 compress() [2/6]

```
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. 560)*	**Error** (p. 106) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 272).

G.61.3.3 compress() [3/6]

```
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 uncompressedData.
outputFile	Location to save compressed file.

Exceptions

Error::ObjectExists *(p. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 272).

G.61.3.4 compress() [4/6]

```
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. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 273).

G.61.3.5 compress() [5/6]

```
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. 453)*	Input file does not exist.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 273).

G.61.3.6 compress() [6/6]

```
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. 453)*	Input file does not exist.
Error::ObjectExists *(p. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 273).

G.61.3.7 decompress() [1/6]

```
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 compressedData.

Returns

Decompressed data.

Exceptions

Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 274).

G.61.3.8 decompress() [2/6]

```
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. 560)*	**Error** (p. 106) in decompression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 275).

G.61.3.9 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. 560)*	**Error** (p. 106) in decompression unit.

Exceptions

Error::ObjectDoesNotExists	Output file already exists.

Implements **BiometricEvaluation::IO::Compressor** (p. 275).

G.61.3.10 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.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	Input file does not exist.
Error::ObjectExists *(p. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 277).

G.61.3.11 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 compressedData.
outputFile	Path to location where decompressed version will be saved.

Exceptions

Error::ObjectExists *(p. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 276).

G.61.3.12 decompress() [6/6]

```
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. 454)*	Output file already exists.
Error::StrategyError *(p. 560)*	**Error** (p. 106) in compression unit.

Implements **BiometricEvaluation::IO::Compressor** (p. 276).

G.61.3.13 operator=()

```
GZip& BiometricEvaluation::IO::GZip::operator= (
            const GZip & other ) [delete]
```

Assignment overload (disabled).
Disabled because **Properties** (p. 482) member of parent cannot be assigned.

Parameters

other	**GZip** (p. 343) to assign.

Returns

lhs **GZip** (p. 343).

G.61.4 Member Data Documentation

G.61.4.1 CHUNK_SIZE

```
const std::string BiometricEvaluation::IO::GZip::CHUNK_SIZE [static]
```

How many bytes to work at a time

G.61.4.2 COMPRESSION_LEVEL

```
const std::string BiometricEvaluation::IO::GZip::COMPRESSION_LEVEL [static]
```

How thorough the compression should be

G.61.4.3 COMPRESSION_METHOD

`const std::string BiometricEvaluation::IO::GZip::COMPRESSION_METHOD [static]`

Which underlying method in the compressor

G.61.4.4 COMPRESSION_STRATEGY

`const std::string BiometricEvaluation::IO::GZip::COMPRESSION_STRATEGY [static]`

Which underlying algorithm to use

G.61.4.5 INPUT_DATA_TYPE

`const std::string BiometricEvaluation::IO::GZip::INPUT_DATA_TYPE [static]`

The type of data being compressed

G.61.4.6 MEMORY_LEVEL

`const std::string BiometricEvaluation::IO::GZip::MEMORY_LEVEL [static]`

How much memory for internal compression state

G.61.4.7 WINDOW_BITS

`const std::string BiometricEvaluation::IO::GZip::WINDOW_BITS [static]`

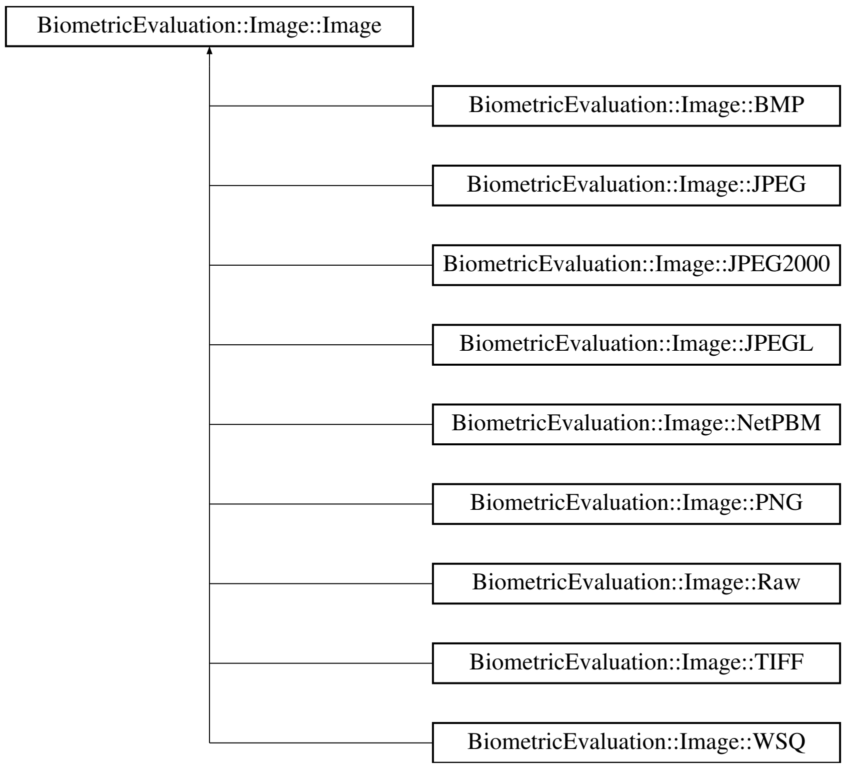
Window size

G.62 BiometricEvaluation::Image::Image Class Reference

Represent attributes common to all images.

`#include <be_image_image.h>`

Inheritance diagram for BiometricEvaluation::Image::Image:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::BMP
BiometricEvaluation::Image::JPEG
BiometricEvaluation::Image::JPEG2000
BiometricEvaluation::Image::JPEGL
BiometricEvaluation::Image::NetPBM
BiometricEvaluation::Image::PNG
BiometricEvaluation::Image::Raw
BiometricEvaluation::Image::TIFF
BiometricEvaluation::Image::WSQ

Public Member Functions

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

  Parent constructor for all ***Image*** *(*p. *351) classes.*
- **Image** (const uint8_t ∗data, const uint64_t size, const **CompressionAlgorithm** compression)

  Parent constructor for all ***Image*** *(*p. *351) 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.

Static Public Member Functions

- static uint64_t **valueInColorspace** (uint64_t color, uint64_t maxColorValue, uint8_t depth)

  Calculate an equivalent color value for a color in an alternate colorspace.
- static std::shared_ptr< **Image** > **openImage** (const uint8_t *data, const uint64_t size)

  *Determine the image type of a buffer of image data and create an **Image** (p. 351) object.*
- static std::shared_ptr< **Image** > **openImage** (const **Memory::uint8Array** &data)

  *Determine the image type of a buffer of image data and create an **Image** (p. 351) object.*
- static std::shared_ptr< **Image** > **openImage** (const std::string &path)

  *Determine the image type of an image file and create an **Image** (p. 351) 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. 493) version of an **Image::Image** (p. 351).*

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.

G.62.1 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. 402), etc. Implementations of this abstraction provide the getRawData method to convert image data to 'raw' format.

Image (p. 351) resolution is in pixels per centimeter, and the coordinate system has the origin at the upper left of the image.

G.62.2 Constructor & Destructor Documentation

G.62.2.1 Image() [1/2]

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

Parent constructor for all **Image** (p. 351) classes.

Parameters

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

Exceptions

Error::StrategyError *(p. 560)*	**Error** (p. 106) manipulating data.
Error::StrategyError *(p. 560)*	**Error** (p. 106) while creating **Image** (p. 351).

G.62.2.2 Image() [2/2]

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

Parent constructor for all **Image** (p. 351) classes.

Parameters

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

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) manipulating data.
Error::StrategyError *(p. 560)*	**Error** (p. 106) while creating **Image** (p. 351).

G.62.3 Member Function Documentation

G.62.3.1 getBitDepth()

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

Accessor for the number of bits per color component.

Returns

The bit depth of the image (in bits).

G.62.3.2 getColorDepth()

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

Accessor for the color depth of the image in bits.

Returns

The color depth of the image (bit).

G.62.3.3 getCompressionAlgorithm() [1/4]

```
CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm ( ) const
```

Accessor for the CompressionAlgorithm of the image.

Returns

Type of compression used on the data that will be returned from **getData()** (p. 357).

G.62.3.4 getCompressionAlgorithm() [2/4]

```
static CompressionAlgorithm BiometricEvaluation::Image::Image::getCompressionAlgorithm (
            const uint8_t * data,
            const uint64_t size ) [static]
```

Determine the compression algorithm of a buffer of image data.

Parameters

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

Returns

Compression algorithm used in the buffer.

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation **Framework** (p. 115) is found.

G.62.3.5 getCompressionAlgorithm() [3/4]

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

Determine the compression algorithm of a buffer of image data.

Parameters

in	*data*	The image data.

Returns

Compression algorithm used in the buffer.

Attention

CompressionAlgorithm::None is returned if no compression algorithm known to the Biometric Evaluation **Framework** (p. 115) is found.

G.62.3.6 getCompressionAlgorithm() [4/4]

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

Determine the compression algorithm of a file.

Parameters

in	*path*	Path to file.

Returns

Compression algorithm used in the file.

Exceptions

Error::ObjectDoesNotExist (p. *453*)	path does not exist.
Error::StrategyError (p. *560*)	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. 115) is found.

G.62.3.7 getData()

`Memory::uint8Array BiometricEvaluation::Image::Image::getData ( ) const`

Accessor for the image data. The data returned is likely encoded in a specialized format.

Returns

AutoArray holding image data.

G.62.3.8 getDataPointer()

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

Returns

Const pointer to buffer underlying _data.

G.62.3.9 getDataSize()

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

Returns

Size (p. 542) of _data.

G.62.3.10 getDimensions()

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

Accessor for the dimensions of the image in pixels.

Returns

Coordinate (p. 283) object containing dimensions in pixels.

G.62.3.11 getRawData() [1/2]

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

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) decompressing image data.

Implemented in **BiometricEvaluation::Image::NetPBM** (p. 450), **BiometricEvaluation::Image::J↩ PEG** (p. 403), **BiometricEvaluation::Image::BMP** (p. 251), **BiometricEvaluation::Image::JPEG2000** (p. 405), **BiometricEvaluation::Image::Raw** (p. 493), **BiometricEvaluation::Image::JPEGL** (p. 407), **BiometricEvaluation::Image::PNG** (p. 473), **BiometricEvaluation::Image::WSQ** (p. 605), and **Biometric↩ Evaluation::Image::TIFF** (p. 570).

G.62.3.12 getRawData() [2/2]

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

Accessor for the raw image data. The data returned should not be compressed or encoded.

Parameters

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

Returns

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

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) decompressing image data.
Error::ParameterError *(p. 470)*	Propagated from **Image::removeComponents** (p. 121).
Error::StrategyError *(p. 560)*	Propagated from **Image::removeComponents** (p. 121).

G.62.3.13 getRawGrayscaleData()

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

Accessor for decompressed data in grayscale.

Parameters

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

Returns

AutoArray holding raw grayscale image data.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 470)*	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::NetPBM** (p. 450), **BiometricEvaluation::Image::↩ JPEG** (p. 403), **BiometricEvaluation::Image::BMP** (p. 252), **BiometricEvaluation::Image::JPEG2000** (p. 406), **BiometricEvaluation::Image::Raw** (p. 494), **BiometricEvaluation::Image::PNG** (p. 473), **Biometric↩ Evaluation::Image::WSQ** (p. 605), **BiometricEvaluation::Image::TIFF** (p. 570), and **BiometricEvaluation↩ ::Image::JPEGL** (p. 408).

G.62.3.14 getRawImage()

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

Obtain **Image::Raw** (p. 493) version of an **Image::Image** (p. 351).

Parameters

in	*image*	Shared pointer to an **Image::Image** (p. 351).

Returns

Shared pointer to an **Image::Raw** (p. 493) version of `image`.

Note

If `image` is already an **Image::Raw** (p. 493), `image` is returned to avoid a copy.

G.62.3.15 getResolution()

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

Accessor for the resolution of the image.

Returns

Resolution (p. 526) struct

G.62.3.16 hasAlphaChannel()

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

Accessor for the presence of an alpha channel.

Returns

Whether or not an alpha channel is present.

G.62.3.17 openImage() [1/3]

```
static std::shared_ptr< Image> BiometricEvaluation::Image::Image::openImage (
             const uint8_t * data,
             const uint64_t size )  [static]
```

Determine the image type of a buffer of image data and create an **Image** (p. 351) object.

Parameters

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

Returns

Image (p. 351) representation of the input data buffer.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) manipulating data.
Error::StrategyError *(p. 560)*	**Error** (p. 106) while creating **Image** (p. 351).

G.62.3.18 openImage() [2/3]

```
static std::shared_ptr< Image> BiometricEvaluation::Image::Image::openImage (
             const  Memory::uint8Array & data )  [static]
```

Determine the image type of a buffer of image data and create an **Image** (p. 351) object.

Parameters

in	*data*	The image data.

Returns

Image (p. 351) representation of the input data buffer.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) manipulating data.
Error::StrategyError *(p. 560)*	**Error** (p. 106) while creating **Image** (p. 351).

G.62.3.19 openImage() [3/3]

```
static std::shared_ptr< Image> BiometricEvaluation::Image::Image::openImage (
             const std::string & path )  [static]
```

Determine the image type of an image file and create an **Image** (p. 351) object.

Parameters

in	*path*	Path to image data.

Returns

Image (p. 351) representation of the input data buffer.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) manipulating data.
Error::ObjectDoesNotExist *(p. 453)*	No file at specified path.
Error::StrategyError *(p. 560)*	**Error** (p. 106) while creating **Image** (p. 351).

G.62.3.20 setBitDepth()

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

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

Parameters

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

G.62.3.21 setColorDepth()

```
void BiometricEvaluation::Image::Image::setColorDepth (
             const uint32_t colorDepth )  [protected]
```

Mutator for the color depth of the image in bits.

Parameters

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

G.62.3.22 setDimensions()

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

Mutator for the dimensions of the image in pixels.

Parameters

in	*dimensions*	Dimensions of image (pixel).

G.62.3.23 setHasAlphaChannel()

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

Mutator for the presence of an alpha channel.

Parameters

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

G.62.3.24 setResolution()

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

Mutator for the resolution of the image .

Parameters

in	*resolution*	**Resolution** (p. 526) struct.

G.62.3.25 valueInColorspace()

```
static uint64_t BiometricEvaluation::Image::Image::valueInColorspace (
             uint64_t color,
             uint64_t maxColorValue,
             uint8_t depth )  [static]
```

Calculate an equivalent color value for a color in an alternate colorspace.

Parameters

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

Returns

A value equivalent to color in depth-bit space.

G.63 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**

G.63.1 Detailed Description

A structure representing information about the image and extended feature set region.

G.63.2 Member Data Documentation

G.63.2.1 fpp

`FPPPosition BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::fpp`

The Finger/Palm/Plantar Position: Mandatory field.

G.63.2.2 ort

`Orientation BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::ort`

The image orientation. Optional but always present due to default value.

G.63.2.3 plr

`LateralReversal BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::plr`

The possible latent reversal information. Optional.

G.63.2.4 roi

`BiometricEvaluation::Image::ROI BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::roi`

The region of interest: A mandatory field.

G.63.2.5 trv

`TonalReversal BiometricEvaluation::Feature::AN2K11EFS::ImageInfo::trv`

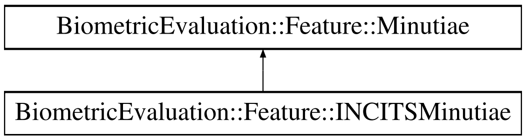
The tonal reversal information. Optional.

G.64 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:


BiometricEvaluation::Feature::Minutiae
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. *438*) *object from its components.*
- **INCITSMinutiae** ()

  Default constructor for an INCITS ***Minutiae*** (p. *438*) *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

G.64.1 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.

G.64.2 Constructor & Destructor Documentation

G.64.2.1 INCITSMinutiae()

```
BiometricEvaluation::Feature::INCITSMinutiae::INCITSMinutiae (
             const MinutiaPointSet & mps,
             const RidgeCountItemSet & rcis,
             const CorePointSet & cps,
             const DeltaPointSet & dps )
```

Construct an INCITS **Minutiae** (p. 438) 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.

G.64.3 Member Function Documentation

G.64.3.1 setCorePointSet()

```
void BiometricEvaluation::Feature::INCITSMinutiae::setCorePointSet (
             const CorePointSet & cps )
```

Mutator for the set of core points.

Parameters

in	*cps*	The set of core points.

G.64.3.2 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.

G.64.3.3 setMinutiaPoints()

```
void BiometricEvaluation::Feature::INCITSMinutiae::setMinutiaPoints (
             const MinutiaPointSet & mps )
```

Mutator for the minutiae point set.

Parameters

in	*mps*	The minutiae points.

G.64.3.4 setRidgeCountItems()

```
void BiometricEvaluation::Feature::INCITSMinutiae::setRidgeCountItems (
             const RidgeCountItemSet & rcis )
```

Mutator for the ridge count items.

Parameters

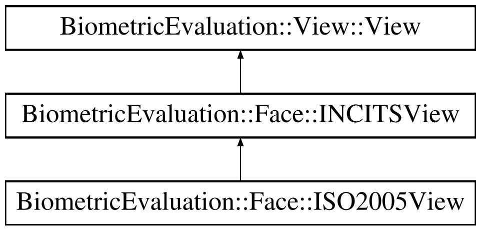
in	*rcis*	The set of ridge count items.

G.65 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:


BiometricEvaluation::View::View
BiometricEvaluation::Face::INCITSView
BiometricEvaluation::Face::ISO2005View

Public Member Functions

- **Face::Gender getGender** () const

  Obtain the gender.
- **Face::EyeColor getEyeColor** () const

  Obtain the eye color.
- **Face::HairColor getHairColor** () const

  Obtain the hair color.
- bool **propertiesConsidered** () const

  Indicate whether properties are specified.
- void **getPropertySet** (**Face::PropertySet** &propertySet) const

Get the set of properties.

- **BiometricEvaluation::Face::Expression getExpression** () const
- void **getFeaturePointSet** (BiometricEvaluation::Feature::MPEGFacePointSet &featurePointSet) const

  Obtain the set of.
- **Face::ImageType getImageType** () const

  Obtain the face image type.
- **Face::ImageDataType getImageDataType** () const

  Obtain the face image data type.
- **Face::PoseAngle getPoseAngle** () const

  Obtain the face pose angle.
- **Face::ColorSpace getColorSpace** () const

  Obtain the color space.
- **Face::SourceType getSourceType** () const

  Obtain the source type.
- uint16_t **getDeviceType** () const

  Obtain the device type.

Protected Member Functions

- **INCITSView** (const std::string &filename, const uint32_t viewNumber)

  Construct the common components of an INCITS face view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an INCITS face view from a record contained in a buffer.
- **Memory::uint8Array** const & **getFIDData** () const

  Obtain a reference to the face image record data buffer.
- virtual void **readHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf, const uint32_↩
  t 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.

Static Protected Attributes

- static const uint32_t **ISO2005_STANDARD** = 1
- static const uint32_t **BASE_FORMAT_ID** = 0x46414300

G.65.1 Detailed Description

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

A base **Face::INCITSView** (p. 367) 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.

G.65.2 Constructor & Destructor Documentation

G.65.2.1 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.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError *(p. 293)*	Invalid record format.
Error::FileError *(p. 312)*	Could not open or read from file.

G.65.2.2 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. 293)*	Invalid record format.

G.65.3 Member Function Documentation

G.65.3.1 getColorSpace()

```
Face::ColorSpace BiometricEvaluation::Face::INCITSView::getColorSpace ( ) const
```

Obtain the color space.

Returns

The color space code.

G.65.3.2 getDeviceType()

`uint16_t BiometricEvaluation::Face::INCITSView::getDeviceType ( ) const`

Obtain the device type.

Returns

The device type vendor code.

G.65.3.3 getEyeColor()

`Face::EyeColor BiometricEvaluation::Face::INCITSView::getEyeColor ( ) const`

Obtain the eye color.

Returns

The eye color code.

G.65.3.4 getFeaturePointSet()

```
void BiometricEvaluation::Face::INCITSView::getFeaturePointSet (
            BiometricEvaluation::Feature::MPEGFacePointSet & featurePointSet ) const
```

Obtain the set of.

Parameters

out	*featurePointSet*	The set of feature points.

G.65.3.5 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.

G.65.3.6 getGender()

`Face::Gender BiometricEvaluation::Face::INCITSView::getGender ( ) const`

Obtain the gender.

Returns

The gender code.

G.65.3.7 getHairColor()

`Face::HairColor BiometricEvaluation::Face::INCITSView::getHairColor ( ) const`

Obtain the hair color.

Returns

The hair color code.

G.65.3.8 getImageDataType()

`Face::ImageDataType BiometricEvaluation::Face::INCITSView::getImageDataType ( ) const`

Obtain the face image data type.

Returns

The image data type.

G.65.3.9 getImageType()

`Face::ImageType BiometricEvaluation::Face::INCITSView::getImageType ( ) const`

Obtain the face image type.

Returns

The image type.

G.65.3.10 getPoseAngle()

`Face::PoseAngle BiometricEvaluation::Face::INCITSView::getPoseAngle ( ) const`

Obtain the face pose angle.

Returns

The pose angle.

G.65.3.11 getPropertySet()

```
void BiometricEvaluation::Face::INCITSView::getPropertySet (
              Face::PropertySet & propertySet ) const
```

Get the set of properties.

Returns

The set of properties.

G.65.3.12 getSourceType()

```
Face::SourceType BiometricEvaluation::Face::INCITSView::getSourceType ( ) const
```

Obtain the source type.

Returns

The source type code.

G.65.3.13 propertiesConsidered()

```
bool BiometricEvaluation::Face::INCITSView::propertiesConsidered ( ) const
```

Indicate whether properties are specified.

Returns

true if properties are specified, false otherwise.

G.65.3.14 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. 107) 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.

G.65.3.15 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

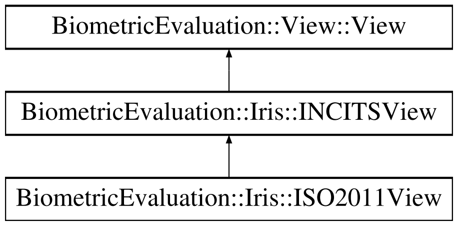
ParameterError	The formatStandard parameter is incorrect.
DataError	The INCITS record has invalid or missing data.

G.66 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:


BiometricEvaluation::View::View
BiometricEvaluation::Iris::INCITSView
BiometricEvaluation::Iris::ISO2011View

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, **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 &iris←CenterLargestX, uint16_t &irisCenterLargestY, uint16_t &irisDiameterSmallest, uint16_t &irisDiameter←Largest)

  Obtain the iris center information. COORDINATE_UNDEF may be returned for any of the out parameters.

Static Public Attributes

- static const uint16_t **RANGE_UNASSIGNED** = 0
- static const uint16_t **RANGE_FAILED** = 1
- static const uint16_t **RANGE_OVERFLOW** = 65535
- static const uint16_t **ROLL_ANGLE_UNDEF** = 65535
- static const uint16_t **ROLL_UNCERTAIN_UNDEF** = 65535
- static const uint16_t **COORDINATE_UNDEF** = 0

Protected Member Functions

- **INCITSView** (const std::string &filename, const uint32_t viewNumber)

  Construct the common components of an INCITS iris view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an INCITS iris view from a record contained in a buffer.
- **Memory::uint8Array** const & **getIIRData** () const

  Obtain a reference to the iris image record data buffer.
- virtual void **readHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf, const uint32_←t formatStandard)

  Read the common iris image record header from an INCITS record, excepting the format identifier and version number data items.
- virtual void **readIrisView** (**Memory::IndexedBuffer** &buf)

  Read the common iris representation information from an INCITS record.

Static Protected Attributes

- static const uint32_t **ISO2011_STANDARD** = 1
- static const uint32_t **BASE_FORMAT_ID** = 0x49495200
- static const uint8_t **CAPTURE_DATE_LENGTH** = 9

G.66.1 Detailed Description

A class to represent single iris view and derived information.

A base **Iris::INCITSView** (p. 373) 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.

G.66.2 Constructor & Destructor Documentation

G.66.2.1 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.
in	*viewNumber*	The eye number to use.

Exceptions

Error::DataError *(p. 293)*	Invalid record format.
Error::FileError *(p. 312)*	Could not open or read from file.

G.66.2.2 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. 293)*	Invalid record format.

G.66.3 Member Function Documentation

G.66.3.1 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.

G.66.3.2 getCaptureDateString()

`std::string BiometricEvaluation::Iris::INCITSView::getCaptureDateString ( ) const`

Obtain the capture date as a string.

Returns

The capture data and time.

G.66.3.3 getCaptureDeviceTechnology()

`Iris::CaptureDeviceTechnology BiometricEvaluation::Iris::INCITSView::getCaptureDeviceTechnology ( ) const`

Obtain the capture device technology.

Returns

The capture device technology identifer.

G.66.3.4 getCaptureDeviceType()

`uint16_t BiometricEvaluation::Iris::INCITSView::getCaptureDeviceType ( ) const`

Obtain the capture device type.

Returns

The capture device type ID.

G.66.3.5 getCaptureDeviceVendor()

`uint16_t BiometricEvaluation::Iris::INCITSView::getCaptureDeviceVendor ( ) const`

Obtain the capture device vendor.

Returns

The capture device vendor ID.

G.66.3.6 getCertificationFlag()

```
uint8_t BiometricEvaluation::Iris::INCITSView::getCertificationFlag ( ) const
```

Obtain the certification flag.

Returns

The certification flag.

G.66.3.7 getEyeLabel()

```
Iris::EyeLabel BiometricEvaluation::Iris::INCITSView::getEyeLabel ( ) const
```

Obtain the eye label type.

Returns

The eye label.

G.66.3.8 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.

G.66.3.9 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.

G.66.3.10 getImageType()

```
Iris::ImageType BiometricEvaluation::Iris::INCITSView::getImageType ( ) const
```

Obtain the iris image type.

Returns

The image type.

G.66.3.11 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.
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.

G.66.3.12 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.

G.66.3.13 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.

G.66.3.14 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.
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.

G.66.3.15 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. 136) 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. 136) Representation.

Exceptions

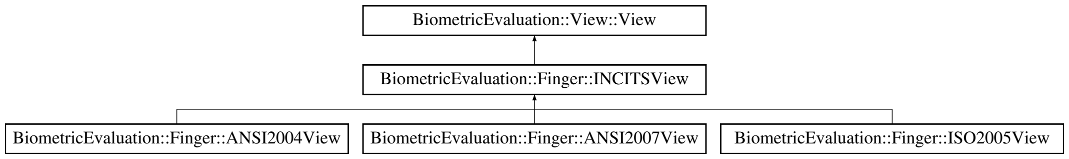
DataError	The INCITS record has invalid or missing data.

G.67 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:


BiometricEvaluation::View::View
BiometricEvaluation::Finger::INCITSView
BiometricEvaluation::Finger::ANSI2004View
BiometricEvaluation::Finger::ANSI2007View
BiometricEvaluation::Finger::ISO2005View

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. 364) item.*
- void **setMinutiaeReservedData** (const std::vector< uint8_t > &reservedBits)

  Mutator for the FMD reserved bits vector.

Static Public Member Functions

- static **Finger::Position convertPosition** (int incitsFGP)

  Convert a finger postion code from an INCITS finger record to the common code.
- static **Finger::Impression convertImpression** (int incitsIMP)

  Convert a impression type code from an INCITS finger record to the common code.

Protected Member Functions

- **INCITSView** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view↩ Number)

  Construct the common components of an INCITS finger view from records contained in files.
- **INCITSView** (const **Memory::uint8Array** &fmrBuffer, const **Memory::uint8Array** &firBuffer, const uint32_t viewNumber)

  Construct an INCITS finger view from records contained in buffers.
- **Memory::uint8Array** const & **getFMRData** () const

  Obtain a reference to the finger minutiae record data buffer.
- **Memory::uint8Array** const & **getFIRData** () const

  Obtain a reference to the finger image record data buffer.
- void **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.

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

G.67.1 Detailed Description

A class to represent single finger view and derived information.

A base **Finger::INCITSView** (p. 379) 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.

G.67.2 Constructor & Destructor Documentation

G.67.2.1 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.
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. 293)*	Invalid record format.
Error::FileError *(p. 312)*	Could not open or read from file.

G.67.2.2 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.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError *(p. 293)*	Invalid record format.

G.67.3 Member Function Documentation

G.67.3.1 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. 293)*	The impression type code is invalid.

Returns

The finger impression type code in common notation.

G.67.3.2 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. 293)*	The position code is invalid.

Returns

The finger position code in common notation.

G.67.3.3 getCaptureEquipmentID()

`uint16_t BiometricEvaluation::Finger::INCITSView::getCaptureEquipmentID ( ) const`

Obtain the capture equipment identifier.

Returns

The equipment ID.

G.67.3.4 getEDBLength()

`uint16_t BiometricEvaluation::Finger::INCITSView::getEDBLength ( ) const`

Returns

Length of extended data block, as recorded in the record.

G.67.3.5 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.

G.67.3.6 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.

G.67.3.7 getFMRReservedByte()

`uint8_t BiometricEvaluation::Finger::INCITSView::getFMRReservedByte ( ) const`

Returns

Reserved byte from FMR header.

G.67.3.8 getImpressionType()

`Finger::Impression BiometricEvaluation::Finger::INCITSView::getImpressionType ( ) const`

Obtain the finger impression code.

Returns

The finger impression code.

G.67.3.9 getMinutiaeReservedData()

`std::vector<uint8_t> BiometricEvaluation::Finger::INCITSView::getMinutiaeReservedData ( ) const`

Returns

FMD reserved bits.

Note

Only lowest 2 bits are relevant.

G.67.3.10 getNumFingerViews()

`uint8_t BiometricEvaluation::Finger::INCITSView::getNumFingerViews ( ) const`

Returns

Number of finger views, as recorded in the record.

G.67.3.11 getPosition()

`Finger::Position BiometricEvaluation::Finger::INCITSView::getPosition ( ) const`

Obtain the finger position.

Returns

The finger position.

G.67.3.12 getProductIDOwner()

`uint16_t BiometricEvaluation::Finger::INCITSView::getProductIDOwner ( ) const [inline]`

Obtain the CBEFF product identifier owner.

Returns

CBEFF product identifier owner.

G.67.3.13 getProductIDType()

`uint16_t BiometricEvaluation::Finger::INCITSView::getProductIDType ( ) const [inline]`

Obtain the CBEFF product identifier type.

Returns

CBEFF product identifier type.

G.67.3.14 getQuality()

```
uint32_t BiometricEvaluation::Finger::INCITSView::getQuality ( ) const
```

Obtain the finger quality value.

Returns

The finger quality value.

G.67.3.15 getRecordLength()

```
uint32_t BiometricEvaluation::Finger::INCITSView::getRecordLength ( ) const
```

Returns

Length of record, as recorded in the record.

G.67.3.16 getViewNumber()

```
uint32_t BiometricEvaluation::Finger::INCITSView::getViewNumber ( ) const
```

Returns

View (p. 162) number, as recorded in the record.

G.67.3.17 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.

G.67.3.18 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::ANSI2007View** (p. 215), **BiometricEvaluation::Finger↩ ::ISO2005View** (p. 400), and **BiometricEvaluation::Finger::ANSI2004View** (p. 212).

G.67.3.19 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.

G.67.3.20 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.
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.

G.67.3.21 readFVMR()

```
void BiometricEvaluation::Finger::INCITSView::readFVMR (
              Memory::IndexedBuffer & buf )  [protected]
```

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

A **Finger** (p. 113) **View** (p. 162) 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.

G.67.3.22 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.

G.67.3.23 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.

G.67.3.24 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.

G.67.3.25 setCaptureEquipmentID()

```
void BiometricEvaluation::Finger::INCITSView::setCaptureEquipmentID (
             uint16_t id )  [protected]
```

Mutator for the equipment ID.

Parameters

in	*id*	The equipment ID value.

G.67.3.26 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.
in	*type*	The CBEFF ID of the product type.

G.67.3.27 setImpressionType()

```
void BiometricEvaluation::Finger::INCITSView::setImpressionType (
             const  Finger::Impression & impression )  [protected]
```

Mutator for the impression type.

Parameters

in	*impression*	The finger impression type code.

G.67.3.28 setMinutiaeData()

```
void BiometricEvaluation::Finger::INCITSView::setMinutiaeData (
             const Feature::INCITSMinutiae & fmd )
```

Mutator for the **Feature::INCITSMinutiae** (p. 364) item.

Parameters

in	*fmd*	The minutiae data object.

G.67.3.29 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.

G.67.3.30 setPosition()

```
void BiometricEvaluation::Finger::INCITSView::setPosition (
             const Finger::Position & position )  [protected]
```

Mutator for the position.

Parameters

in	*position*	The finger position.

G.67.3.31 setQuality()

```
void BiometricEvaluation::Finger::INCITSView::setQuality (
             uint32_t quality )  [protected]
```

Mutator for the finger quality value.

Parameters

in	*quality*	The quality value.

G.67.3.32 setViewNumber()

```
void BiometricEvaluation::Finger::INCITSView::setViewNumber (
            uint32_t viewNumber )  [protected]
```

Mutator for the finger view number.

Parameters

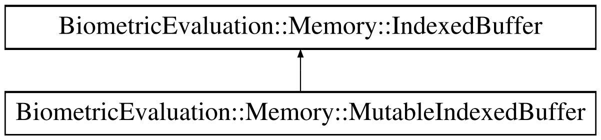
in	*viewNumber*	The view number value.

G.68 BiometricEvaluation::Memory::IndexedBuffer Class Reference

Wrap a memory buffer with an index.

`#include <be_memory_indexedbuffer.h>`

Inheritance diagram for BiometricEvaluation::Memory::IndexedBuffer:


BiometricEvaluation::Memory::IndexedBuffer
BiometricEvaluation::Memory::MutableIndexedBuffer

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

G.68.1 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.

G.68.2 Constructor & Destructor Documentation

G.68.2.1 IndexedBuffer() `[1/4]`

```
BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer ( )
```

Wrap a nullptr buffer.

G.68.2.2 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.

G.68.2.3 IndexedBuffer() `[3/4]`

```
BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (
            const uint8Array & aa )
```

Wrap an existing uint8Array.

Parameters

aa	uint8Array to wrap.

G.68.2.4 IndexedBuffer() [4/4]

```
BiometricEvaluation::Memory::IndexedBuffer::IndexedBuffer (
            const  IndexedBuffer & copy )  [default]
```

Copy constructor (default).

G.68.2.5 ∼IndexedBuffer()

```
virtual BiometricEvaluation::Memory::IndexedBuffer::∼IndexedBuffer ( )  [virtual], [default]
```

Destructor (default).

G.68.3 Member Function Documentation

G.68.3.1 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. 443).

G.68.3.2 getIndex()

```
uint32_t BiometricEvaluation::Memory::IndexedBuffer::getIndex ( ) const
```

Obtain the current index into the buffer.

Returns

The current buffer index.

Note

When **getIndex()** (p. 393) == **getSize()** (p. 393), the buffer is exhausted from scanning.

G.68.3.3 getSize()

```
uint32_t BiometricEvaluation::Memory::IndexedBuffer::getSize ( ) const
```

Obtain the current size of the buffer.

Returns

The current buffer size.

G.68.3.4 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. *293*)	The buffer is exhausted.

Returns

The number of elements copied.

G.68.3.5 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. *293*)	The buffer is exhausted.

G.68.3.6 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. *293*)	The buffer is exhausted.

G.68.3.7 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. 293)*	The buffer is exhausted.

G.68.3.8 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. 293)*	The buffer is exhausted.

G.68.3.9 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. 293)*	The buffer is exhausted.

G.68.3.10 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. 293)	The buffer is exhausted.

G.68.3.11 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

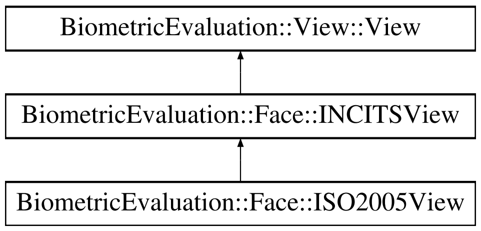
Error::ParameterError (p. 470)	The index parameter is too large.

G.69 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:


BiometricEvaluation::View::View
BiometricEvaluation::Face::INCITSView
BiometricEvaluation::Face::ISO2005View

Public Member Functions

- **ISO2005View** ()

  *Construct an empty ISO2005 **Face** (p. 107) **Image** (p. 118) Data record.*
- **ISO2005View** (const std::string &filename, const uint32_t viewNumber)

  Construct an ISO 2005 face view from the named file.
- **ISO2005View** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an ISO 2005 face view from a record contained in a buffer.

Protected Member Functions

- void **readISOHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf)

  Read the face image data record header from an ISO 2005 record.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x30313000

G.69.1 Detailed Description

A class to represent single face view and derived information.

A base **Face::ISO2005View** (p. 396) class represents an ISO 2005 face image data view.

G.69.2 Constructor & Destructor Documentation

G.69.2.1 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.
in	*viewNumber*	The facial information instance to read.

Exceptions

Error::DataError (p. 293)	Invalid record format.
Error::FileError (p. 312)	Could not open or read from file.

G.69.2.2 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.

Parameters

in	*viewNumber*	The facial information instance to read.

Exceptions

Error::DataError *(p. 293)*	Invalid record format.

G.69.3 Member Function Documentation

G.69.3.1 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

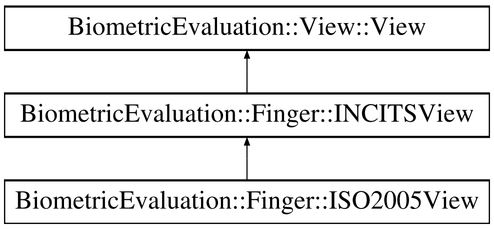
DataError	The record has invalid or missing data.

G.70 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:


BiometricEvaluation::View::View
BiometricEvaluation::Finger::INCITSView
BiometricEvaluation::Finger::ISO2005View

Public Member Functions

- **ISO2005View** (const std::string &fmrFilename, const std::string &firFilename, const uint32_t view↩ Number)

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.

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.

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x20323000

Additional Inherited Members

G.70.1 Detailed Description

A class to represent single finger view and derived information.

A **Finger::ISO2005View** (p. 398) object represents a finger view from a ISO/IEC-2005 **Finger** (p. 113) Minutiae Record.

G.70.2 Constructor & Destructor Documentation

G.70.2.1 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.
in	*firFilename*	The name of the file containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

G.70.2.2 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.
in	*firBuffer*	The buffer containing the complete finger image record.
in	*viewNumber*	The finger view number to use.

Exceptions

Error::DataError *(p. 293)*	Invalid record format.

G.70.3 Member Function Documentation

G.70.3.1 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.

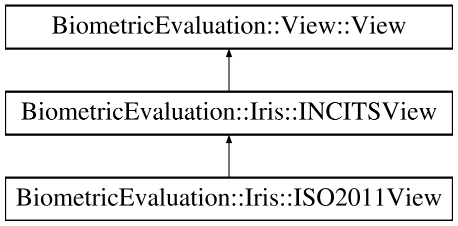
Implements **BiometricEvaluation::Finger::INCITSView** (p. 386).

G.71 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:


BiometricEvaluation::View::View
BiometricEvaluation::Iris::INCITSView
BiometricEvaluation::Iris::ISO2011View

Public Member Functions

- **ISO2011View** ()

  Construct an empty ISO 2011 iris view.
- **ISO2011View** (const std::string &filename, const uint32_t viewNumber)

  Construct an ISO 2011 iris view from the named file.
- **ISO2011View** (const **Memory::uint8Array** &buffer, const uint32_t viewNumber)

  Construct an ISO 2011 iris view from a record contained in a buffer.

Protected Member Functions

- void **readISOHeader** (**BiometricEvaluation::Memory::IndexedBuffer** &buf)

Static Protected Attributes

- static const uint32_t **BASE_SPEC_VERSION** = 0x30323000

Additional Inherited Members

G.71.1 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.

G.71.2 Constructor & Destructor Documentation

G.71.2.1 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. 293)*	Invalid record format.
Error::FileError *(p. 312)*	Could not open or read from file.

G.71.2.2 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

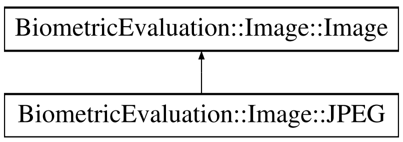
Error::DataError *(p. 293)*	Invalid record format.

G.72 BiometricEvaluation::Image::JPEG Class Reference

A JPEG-encoded image.

`#include <be_image_jpeg.h>`

Inheritance diagram for BiometricEvaluation::Image::JPEG:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::JPEG

Public Member Functions

- **JPEG** (const uint8_t ∗data, const uint64_t size)
- **JPEG** (const **Memory::uint8Array** &data)
- **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.

Static Public Member Functions

- static bool **isJPEG** (const uint8_t ∗data, uint64_t size)
- static int **getc_skip_marker_segment** (const unsigned short marker, unsigned char ∗∗cbufptr, unsigned char ∗ebufptr)

Additional Inherited Members

G.72.1 Detailed Description

A JPEG-encoded image.

G.72.2 Member Function Documentation

G.72.2.1 getRawData()

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

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 357).

G.72.2.2 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. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 470)*	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. 358).

G.72.2.3 isJPEG()

```
static bool BiometricEvaluation::Image::JPEG::isJPEG (
            const uint8_t * data,
            uint64_t size ) [static]
```

Whether or not data is a Lossy **JPEG** (p. 402) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

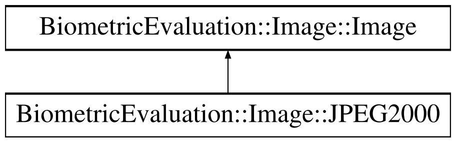
> true if data appears to be a Lossy **JPEG** (p. 402) image, false otherwise

G.73 BiometricEvaluation::Image::JPEG2000 Class Reference

A JPEG-2000-encoded image.

```
#include <be_image_jpeg2000.h>
```

Inheritance diagram for BiometricEvaluation::Image::JPEG2000:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::JPEG2000

Public Member Functions

- **JPEG2000** (const uint8_t ∗data, const uint64_t size, const int8_t codecFormat=2)

  Create a new ***JPEG2000*** *(p. 404) object.*
- **JPEG2000** (const **Memory::uint8Array** &data)
- **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.

Static Public Member Functions

- static bool **isJPEG2000** (const uint8_t ∗data, uint64_t size)

Additional Inherited Members

G.73.1 Detailed Description

A JPEG-2000-encoded image.

G.73.2 Constructor & Destructor Documentation

G.73.2.1 JPEG2000()

```
BiometricEvaluation::Image::JPEG2000::JPEG2000 (
             const uint8_t * data,
             const uint64_t size,
             const int8_t codecFormat = 2 )
```

Create a new **JPEG2000** (p. 404) object.

Parameters

in	*data*	The image data.
in	*size*	The size of the image data, in bytes.
in	*codec*	The OPJ_CODEC_FORMAT used to encode data.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) manipulating data.
Error::StrategyError *(p. 560)*	**Error** (p. 106) while creating **Image** (p. 351).

G.73.3 Member Function Documentation

G.73.3.1 getRawData()

```
Memory::uint8Array BiometricEvaluation::Image::JPEG2000::getRawData ( ) const  [virtual]
```

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 357).

G.73.3.2 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. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 470)*	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. 358).

G.73.3.3 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

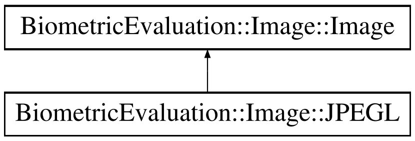
true if data appears to be a JPEG-2000 image, false otherwise.

G.74 BiometricEvaluation::Image::JPEGL Class Reference

A Lossless JPEG-encoded image.

`#include <be_image_jpegl.h>`

Inheritance diagram for BiometricEvaluation::Image::JPEGL:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::JPEGL

Public Member Functions

- **JPEGL** (const uint8_t ∗data, const uint64_t size)
- **JPEGL** (const **Memory::uint8Array** &data)
- **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.

Static Public Member Functions

- static bool **isJPEGL** (const uint8_t ∗data, uint64_t size)

Additional Inherited Members

G.74.1 Detailed Description

A Lossless JPEG-encoded image.

G.74.2 Member Function Documentation

G.74.2.1 getRawData()

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

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 357).

G.74.2.2 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. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 470)*	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. 358).

G.74.2.3 isJPEGL()

```
static bool BiometricEvaluation::Image::JPEGL::isJPEGL (
            const uint8_t * data,
            uint64_t size ) [static]
```

Whether or not data is a Lossless **JPEG** (p. 402) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

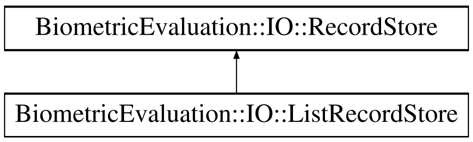
true if data appears to be a Lossless **JPEG** (p. 402) image, false otherwise.

G.75 BiometricEvaluation::IO::ListRecordStore Class Reference

RecordStore (p. 500) that reads a list of keys from a text file, and retrieves the data from another **RecordStore** (p. 500).

`#include <be_io_listrecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ListRecordStore:


BiometricEvaluation::IO::RecordStore
BiometricEvaluation::IO::ListRecordStore

Public Member Functions

- **ListRecordStore** (const std::string &pathname)
- ∼**ListRecordStore** ()
- void **insert** (const std::string &key, const void ∗const data, const uint64_t size) 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. 500), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 500), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- void **move** (const std::string &pathname) override

  *Move the **RecordStore** (p. 500).*
- 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

Additional Inherited Members

G.75.1 Detailed Description

RecordStore (p. 500) that reads a list of keys from a text file, and retrieves the data from another **RecordStore** (p. 500).

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. 500) 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. 500); see example below.

Second, create a file called 'KeyList.txt' in the **RecordStore** (p. 500) directory containing a list of keys, one per line.

ListRecordStores can also be created and modified with versions of rstool(1) from 2013 or later.

Example .rscontrol.prop file: Count = 10 Description = Search records for SDK TESTSDK Name = Test↩ LRS Type = List Source Record Store = /Users/wsalamon/sandbox/SD29.rs

Note

List RecordStores must be opened read-only.

G.75.2 Constructor & Destructor Documentation

G.75.2.1 ListRecordStore()

```
BiometricEvaluation::IO::ListRecordStore::ListRecordStore (
            const std::string & pathname )
```

Constructor, always opening read-only

G.75.2.2 ∼ListRecordStore()

```
BiometricEvaluation::IO::ListRecordStore::∼ListRecordStore ( )
```

Destructor

G.75.3 Member Function Documentation

G.75.3.1 changeDescription()

```
void BiometricEvaluation::IO::ListRecordStore::changeDescription (
            const std::string & description ) [override], [virtual]
```

Change the description of the **RecordStore** (p. 500).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 502).

G.75.3.2 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.75.3.3 getCount()

```
unsigned int BiometricEvaluation::IO::ListRecordStore::getCount ( ) const  [override], [virtual]
```

Obtain the number of items in the **RecordStore** (p. 500).

Returns

The number of items in the **RecordStore** (p. 500).

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.75.3.4 getDescription()

```
std::string BiometricEvaluation::IO::ListRecordStore::getDescription ( ) const  [override], [virtual]
```

Obtain a textual description of the **RecordStore** (p. 500).

Returns

The **RecordStore** (p. 500)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.75.3.5 getPathname()

```
std::string BiometricEvaluation::IO::ListRecordStore::getPathname ( ) const  [override], [virtual]
```

Return the path name of the **RecordStore** (p. 500).

Returns

Where in the file system the **RecordStore** (p. 500) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.75.3.6 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. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.75.3.7 insert()

```
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. 454)*	A record with the given key is already present.
Error::StrategyError *(p. 560)*	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.75.3.8 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.75.3.9 move()

```
void BiometricEvaluation::IO::ListRecordStore::move (
            const std::string & pathname )  [override], [virtual]
```

Move the **RecordStore** (p. 500).
The **RecordStore** (p. 500) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 507).

G.75.3.10 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 508).

G.75.3.11 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 509).

G.75.3.12 replace()

```
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. 500).

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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Reimplemented from **BiometricEvaluation::IO::RecordStore** (p. 510).

G.75.3.13 sequence()

```
RecordStore::Record BiometricEvaluation::IO::ListRecordStore::sequence (
             int cursor =  BE_RECSTORE_SEQ_NEXT )  [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.75.3.14 sequenceKey()

```
std::string BiometricEvaluation::IO::ListRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT ) [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.75.3.15 setCursorAtKey()

```
void BiometricEvaluation::IO::ListRecordStore::setCursorAtKey (
             const std::string & key )  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 500), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 414).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 414).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.75.3.16 sync()

```
void BiometricEvaluation::IO::ListRecordStore::sync ( ) const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

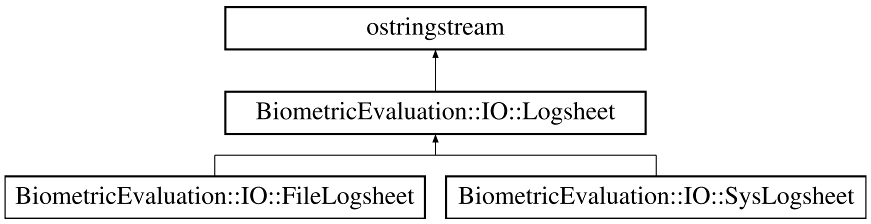
Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.76 BiometricEvaluation::IO::Logsheet Class Reference

A class to represent a logging mechanism.

`#include <be_io_logsheet.h>`

Inheritance diagram for BiometricEvaluation::IO::Logsheet:


ostringstream
BiometricEvaluation::IO::Logsheet
BiometricEvaluation::IO::FileLogsheet
BiometricEvaluation::IO::SysLogsheet

Public Types

- enum **Kind** { **Kind::Null**, **Kind::File**, **Kind::Syslog** }

Public Member Functions

- **Logsheet** ()

  *Create a **Logsheet** (p. 416) that has no backing store. A log entry is maintained, but cannot be permanently stored. This is the Null **Logsheet** (p. 416).*
- 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. 416) 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. 416) 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.

G.76.1 Detailed Description

A class to represent a logging mechanism.

A **Logsheet** (p. 416) is a string stream, so applications can write into the stream as a staging area using the << operator, then start a new entry by calling **newEntry()** (p. 421). Entries in the log are prefixed with an entry number, which is incremented when the entry is written (either by directly calling **write()** (p. 423), or calling **newEntry()** (p. 421)).

How the log data is stored is implemented by subclasses of **Logsheet** (p. 416).

Note

By default, the entries in the **Logsheet** (p. 416) 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. 423), 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. 416) won't check for that condition, but any existing **Logsheet** (p. 416) that is re-opened for append may have an incorrect starting entry number.

G.76.2 Member Enumeration Documentation

G.76.2.1 Kind

`enum BiometricEvaluation::IO::Logsheet::Kind [strong]`

Enumerator

Null	No backing store log sheet
File	File-based log sheet
Syslog	Syslog daemon backing store

G.76.3 Constructor & Destructor Documentation

G.76.3.1 ∼Logsheet()

```
virtual BiometricEvaluation::IO::Logsheet::∼Logsheet ( )  [virtual]
```

Destructor

G.76.4 Member Function Documentation

G.76.4.1 getAutoSync()

```
bool BiometricEvaluation::IO::Logsheet::getAutoSync ( ) const
```

Return the current auto-sync state.

Returns

true if auto-sync is on, false otherwise.

G.76.4.2 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.

G.76.4.3 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.

G.76.4.4 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.

G.76.4.5 getCurrentEntryNumber()

```
uint32_t BiometricEvaluation::IO::Logsheet::getCurrentEntryNumber ( ) const
```

Obtain the current entry number.

Returns

The current entry number.

G.76.4.6 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.

G.76.4.7 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.

G.76.4.8 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. 416) type.

Parameters

in	*url*	The unform resource locator of the **Logsheet** (p. 416).

Returns

The type of **Logsheet** (p. 416) represented by the URL.

Exceptions

Error::ParameterError (p. *470*)	The URL scheme is missing or invalid.

G.76.4.9 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.

G.76.4.10 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.

G.76.4.11 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.

G.76.4.12 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. 560)*	An error occurred when using the underlying backing store.

G.76.4.13 resetCurrentEntry()

```
void BiometricEvaluation::IO::Logsheet::resetCurrentEntry ( )
```

Reset the current entry buffer to the beginning.

G.76.4.14 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 **newEntry()** (p. 421) is or **write()** (p. 423) is called. When false, **sync()** (p. 423) must be called to force a write.

G.76.4.15 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.

G.76.4.16 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.

G.76.4.17 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.

G.76.4.18 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. 560)*	An error occurred when using the underlying backing store.

Reimplemented in **BiometricEvaluation::IO::FileLogsheet** (p. 319), and **BiometricEvaluation::IO↩ ::SysLogsheet** (p. 566).

G.76.4.19 trim()

```
static std::string BiometricEvaluation::IO::Logsheet::trim (
             const std::string & entry )  [static]
```

Trim delimiters from **Logsheet** (p. 416) entries.

Works for comments and numbered entries.

Parameters

in	*entry*	The entry to trim.

Returns

Delimiter-less entry.

G.76.4.20 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. 560)*	An error occurred when using the underlying backing store.

Reimplemented in **BiometricEvaluation::IO::FileLogsheet** (p. 319), and **BiometricEvaluation::IO↩ ::SysLogsheet** (p. 566).

G.76.4.21 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. 560)*	An error occurred when using the underlying backing store.

Reimplemented in **BiometricEvaluation::IO::FileLogsheet** (p. 320), and **BiometricEvaluation::IO↩ ::SysLogsheet** (p. 567).

G.76.4.22 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

Parameters

in	*entry*	The text of the debug message.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when logging.

Reimplemented in **BiometricEvaluation::IO::FileLogsheet** (p. 320), and **BiometricEvaluation::IO↩ ::SysLogsheet** (p. 567).

G.76.5 Member Data Documentation

G.76.5.1 CommentDelimiter

`const char BiometricEvaluation::IO::Logsheet::CommentDelimiter = '#'  [static]`

Delimiter for a comment line in the log sheet.

G.76.5.2 DebugDelimiter

`const char BiometricEvaluation::IO::Logsheet::DebugDelimiter = 'D'  [static]`

Delimiter for an debug line in the log sheet.

G.76.5.3 DescriptionTag

`const std::string BiometricEvaluation::IO::Logsheet::DescriptionTag  [static]`

The tag for the description string.

G.76.5.4 EntryDelimiter

`const char BiometricEvaluation::IO::Logsheet::EntryDelimiter = 'E'  [static]`

Delimiter for an entry line in the log sheet.

G.76.5.5 FILEURLSCHEME

`const std::string BiometricEvaluation::IO::Logsheet::FILEURLSCHEME  [static]`

The URL scheme to be used for **FileLogsheet** (p. 315) URL strings.

G.76.5.6 SYSLOGURLSCHEME

`const std::string BiometricEvaluation::IO::Logsheet::SYSLOGURLSCHEME  [static]`

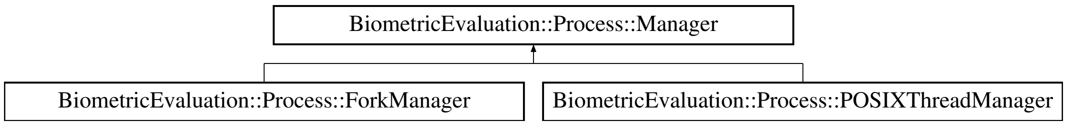
The URL scheme to be used for **SysLogsheet** (p. 563) URL strings.

G.77 BiometricEvaluation::Process::Manager Class Reference

An interface for intranode process management classes.

`#include <be_process_manager.h>`

Inheritance diagram for BiometricEvaluation::Process::Manager:


BiometricEvaluation::Process::Manager
BiometricEvaluation::Process::ForkManager
BiometricEvaluation::Process::POSIXThreadManager

Public Member Functions

- **Manager** ()

  Manager *(p. 426) constructor.*
- virtual std::shared_ptr< **WorkerController** > **addWorker** (std::shared_ptr< **Worker** > worker)=0

  Adds a ***Worker*** *(p. 588) to be managed by this* ***Manager*** *(p. 426).*
- 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. 588)'s work.*
- virtual void **startWorker** (std::shared_ptr< **WorkerController** > worker, bool wait=true, bool communicate=false)=0

  Start a ***Worker*** *(p. 588).*
- 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. 588) 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. 588).*
- virtual bool **getNextMessage** (std::shared_ptr< **WorkerController** > &sender, **Memory::uint8Array** &message, int numSeconds=-1) const

  Obtain a message from a ***Worker*** *(p. 588).*
- virtual void **broadcastMessage** (**Memory::uint8Array** &message) const

  Send one message to all Workers.
- virtual ∼**Manager** ()

  Manager *(p. 426) 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**

G.77.1 Detailed Description

An interface for intranode process management classes.

G.77.2 Member Function Documentation

G.77.2.1 addWorker()

```
virtual std::shared_ptr< WorkerController> BiometricEvaluation::Process::Manager::addWorker (
            std::shared_ptr<  Worker  > worker )  [pure virtual]
```

Adds a **Worker** (p. 588) to be managed by this **Manager** (p. 426).

Parameters

worker	A **Worker** (p. 588) instance to run.

Returns

shared_ptr to worker.

Implemented in **BiometricEvaluation::Process::ForkManager** (p. 332), and **BiometricEvaluation::↩ Process::POSIXThreadManager** (p. 477).

G.77.2.2 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. 560)*	**Error** (p. 106) propagated from the **WorkerController** (p. 595).

G.77.2.3 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. 588).

Parameters

out	*sender*	Reference to a shared pointer of the **WorkerController** (p. 595) that sent the message.
out	*message*	Reference to a buffer to hold the message.
in	*numSeconds*	Number of seconds to wait for a message, or < 0 to block.

Returns

true if there is a message, false otherwise.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	(Unexpected) widowed pipe.
Error::StrategyError *(p. 560)*	**Error** (p. 106) receiving message.

G.77.2.4 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. 560)*	No Workers have started working yet.

G.77.2.5 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. 560)*	No Workers have started working yet.

G.77.2.6 getTotalWorkers()

```
virtual uint32_t BiometricEvaluation::Process::Manager::getTotalWorkers ( ) const  [virtual]
```

Obtain the number of Workers this class is handling.

Returns

Number of Workers.

G.77.2.7 reset()

```
virtual void BiometricEvaluation::Process::Manager::reset ( )  [virtual]
```

Reuse all Workers.

Exceptions

Error::ObjectExists *(p. 454)*	At least one **Worker** (p. 588) is still working.

G.77.2.8 startWorker()

```
virtual void BiometricEvaluation::Process::Manager::startWorker (
            std::shared_ptr< WorkerController > worker,
            bool wait = true,
            bool communicate = false )  [pure virtual]
```

Start a **Worker** (p. 588).

Parameters

	worker	Pointer to a **WorkerController** (p. 595) that is being managed by this **Manager** (p. 426) instance.
	wait	Whether or not to wait for this **Worker** (p. 588) 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. 454)*	worker is already working.

Exceptions

Error::StrategyError *(p. 560)*	worker is not managed by this **Manager** (p. 426) 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. 335), and **BiometricEvaluation::↩ Process::POSIXThreadManager** (p. 478).

G.77.2.9 startWorkers()

```
virtual void BiometricEvaluation::Process::Manager::startWorkers (
            bool wait = true,
            bool communicate = false )  [pure virtual]
```

Begin **Worker** (p. 588)'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. 454)*	At least one **Worker** (p. 588) is already working.
Error::StrategyError *(p. 560)*	Problem starting Workers.

Implemented in **BiometricEvaluation::Process::ForkManager** (p. 335), and **BiometricEvaluation::↩ Process::POSIXThreadManager** (p. 478).

G.77.2.10 stopWorker()

```
virtual void BiometricEvaluation::Process::Manager::stopWorker (
            std::shared_ptr< WorkerController > worker )  [pure virtual]
```

Ask **Worker** (p. 588) to return as soon as possible.

Parameters

worker	Pointer to the **WorkerController** (p. 595) that should be stopped.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	worker is not working.
Error::StrategyError *(p. 560)*	Problem asking worker to stop.

Implemented in **BiometricEvaluation::Process::ForkManager** (p. 336), and **BiometricEvaluation::↩ Process::POSIXThreadManager** (p. 479).

G.77.2.11 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. 588).

Parameters

out	*sender*	Reference to a shared pointer of the **WorkerController** (p. 595) that sent the message.
in,out	*nextFD*	Location to store a pipe that has data to read.
in	*numSeconds*	Number of seconds to wait for a message, or $<$ 0 to block.

Returns

true if there is a **Worker** (p. 588) sending a message false otherwise or if an error occurred.

G.77.2.12 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. 336), and **BiometricEvaluation::↩ Process::POSIXThreadManager** (p. 479).

G.77.3 Member Data Documentation

G.77.3.1 _pendingExit

```
std::vector<std::shared_ptr< WorkerController> > BiometricEvaluation::Process::Manager::_pending↩
Exit  [protected]
```

Workers that are about to exit (stop requested).

G.77.3.2 _workers

```
std::vector<std::shared_ptr< WorkerController> > BiometricEvaluation::Process::Manager::_workers
[protected]
```

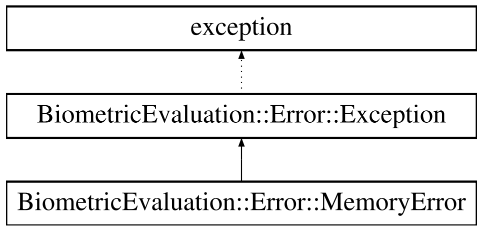
Workers that have been added.

G.78 BiometricEvaluation::Error::MemoryError Class Reference

An error occurred when allocating an object.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::MemoryError:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::MemoryError

Public Member Functions

- **MemoryError** ()
- **MemoryError** (const std::string &info)

G.78.1 Detailed Description

An error occurred when allocating an object.

G.78.2 Constructor & Destructor Documentation

G.78.2.1 MemoryError() [1/2]

```
BiometricEvaluation::Error::MemoryError::MemoryError ( )
```

Construct a **MemoryError** (p. 432) object with the default information string.

G.78.2.2 MemoryError() [2/2]

```
BiometricEvaluation::Error::MemoryError::MemoryError (
                const std::string & info )
```

Construct a **MemoryError** (p. 432) object with an information string appended to the default information string.

G.79 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

G.79.1 Detailed Description

Convenience for asynchronous TCP socket message passing.

G.79.2 Constructor & Destructor Documentation

G.79.2.1 MessageCenter()

```
BiometricEvaluation::Process::MessageCenter::MessageCenter (
             uint32_t port =  MessageCenter::DEFAULT_PORT )
```

Constructor.

Parameters

port	Listening port.

G.79.3 Member Function Documentation

G.79.3.1 disconnectClient()

```
void BiometricEvaluation::Process::MessageCenter::disconnectClient (
             uint32_t clientID )
```

Break the connection with a client.

Parameters

clientID	ID of the client to disconect.

G.79.3.2 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.

G.79.3.3 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.

G.79.3.4 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.

G.79.4 Member Data Documentation

G.79.4.1 CONNECTION_BACKLOG

```
const int BiometricEvaluation::Process::MessageCenter::CONNECTION_BACKLOG = 10  [static]
```

Number of outstanding connections.

G.79.4.2 DEFAULT_PORT

```
const uint16_t BiometricEvaluation::Process::MessageCenter::DEFAULT_PORT = 7899  [static]
```

Default port used for messages.

G.79.4.3 DEFAULT_TIMEOUT

```
const int BiometricEvaluation::Process::MessageCenter::DEFAULT_TIMEOUT = 1  [static]
```

Default number of seconds to wait between polls.

G.79.4.4 MAX_MESSAGE_LENGTH

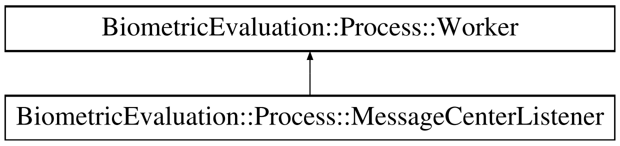
```
const uint64_t BiometricEvaluation::Process::MessageCenter::MAX_MESSAGE_LENGTH = 255  [static]
```

Maximum length of a message.

G.80 BiometricEvaluation::Process::MessageCenterListener Class Reference

`#include <be_process_mclistener.h>`

Inheritance diagram for BiometricEvaluation::Process::MessageCenterListener:


BiometricEvaluation::Process::Worker
BiometricEvaluation::Process::MessageCenterListener

Public Member Functions

- int32_t **workerMain** ()

    The method that will get called to start execution by a ProcessManager.

Static Public Attributes

- static const std::string **PARAM_PORT**

Additional Inherited Members

G.80.1 Detailed Description

Accepts new connections and spawns message receivers.

G.80.2 Member Function Documentation

G.80.2.1 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. 331) object, the implementation of **Process::Worker::workerMain()** (p. 594) 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_FAI↩ LURE. The type and contents of the exception is not maintained.

Implements **BiometricEvaluation::Process::Worker** (p. 594).

G.80.3 Member Data Documentation

G.80.3.1 PARAM_PORT

`const std::string BiometricEvaluation::Process::MessageCenterListener::PARAM_PORT [static]`

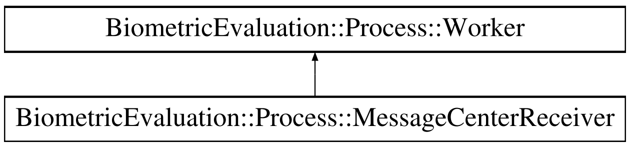
Parameter used to pass port number

G.81 BiometricEvaluation::Process::MessageCenterReceiver Class Reference

Receives message from a client, forwarding to the central **MessageCenter** (p. 432).

`#include <be_process_mcreceiver.h>`

Inheritance diagram for BiometricEvaluation::Process::MessageCenterReceiver:


BiometricEvaluation::Process::Worker
BiometricEvaluation::Process::MessageCenterReceiver

Public Member Functions

- int32_t **workerMain** ()
- **MessageCenterReceiver** ()=default
- ∼**MessageCenterReceiver** ()=default

Static Public Attributes

- static const std::string **PARAM_CLIENT_SOCKET**
- static const std::string **PARAM_CLIENT_ID**
- static const std::string **MSG_DISCONNECT**

Additional Inherited Members

G.81.1 Detailed Description

Receives message from a client, forwarding to the central **MessageCenter** (p. 432).

G.81.2 Constructor & Destructor Documentation

G.81.2.1 MessageCenterReceiver()

`BiometricEvaluation::Process::MessageCenterReceiver::MessageCenterReceiver ( ) [default]`

Default constructor.

G.81.2.2 ∼MessageCenterReceiver()

`BiometricEvaluation::Process::MessageCenterReceiver::∼MessageCenterReceiver ( ) [default]`

Default destructor.

G.81.3 Member Function Documentation

G.81.3.1 workerMain()

`int32_t BiometricEvaluation::Process::MessageCenterReceiver::workerMain ( ) [virtual]`

Receive loop.

Implements **BiometricEvaluation::Process::Worker** (p. 594).

G.81.4 Member Data Documentation

G.81.4.1 MSG_DISCONNECT

`const std::string BiometricEvaluation::Process::MessageCenterReceiver::MSG_DISCONNECT [static]`

Message sent when client should disconnect.

G.81.4.2 PARAM_CLIENT_ID

`const std::string BiometricEvaluation::Process::MessageCenterReceiver::PARAM_CLIENT_ID [static]`

Parameter used to pass an ID to the client.

G.81.4.3 PARAM_CLIENT_SOCKET

`const std::string BiometricEvaluation::Process::MessageCenterReceiver::PARAM_CLIENT_SOCKET [static]`

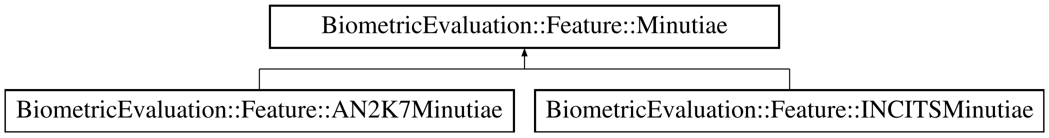
Parameter used to pass client socket FD.

G.82 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:


BiometricEvaluation::Feature::Minutiae
BiometricEvaluation::Feature::AN2K7Minutiae
BiometricEvaluation::Feature::INCITSMinutiae

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.

G.82.1 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.

G.83 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**

G.83.1 Detailed Description

Representation of an extended feature set ridge count info.

G.83.2 Member Data Documentation

G.83.2.1 mia

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mia`

minutia index A

G.83.2.2 mib

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mib`

minutia index B

G.83.2.3 mir

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mir`

ridge count

G.83.2.4 mrn

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mrn`

reference number, optional

G.83.2.5 mrs

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount::mrs`

residual, optional

G.84 BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount↩ Confidence 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**

G.84.1 Detailed Description

Representation of an extended feature set minutiae ridge count confidence item.

G.85 BiometricEvaluation::Feature::AN2K11EFS::MinutiaeRidgeCount↩ Info 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**

G.85.1 Detailed Description

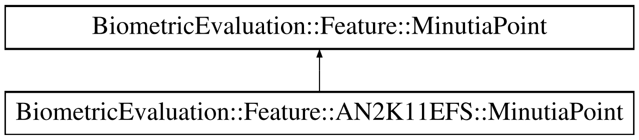
All the ridge count information in one place.

G.86 BiometricEvaluation::Feature::MinutiaPoint Struct Reference

Representation of a finger minutiae data point.

`#include <be_feature_minutiae.h>`

Inheritance diagram for BiometricEvaluation::Feature::MinutiaPoint:


BiometricEvaluation::Feature::MinutiaPoint
BiometricEvaluation::Feature::AN2K11EFS::MinutiaPoint

Public Attributes

- unsigned int **index**
- bool **has_type**
- **MinutiaeType type**
- **Image::Coordinate coordinate**
- unsigned int **theta**
- bool **has_quality**
- unsigned int **quality**

G.86.1 Detailed Description

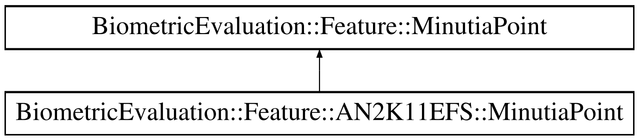
Representation of a finger minutiae data point.

G.87 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:


BiometricEvaluation::Feature::MinutiaPoint
BiometricEvaluation::Feature::AN2K11EFS::MinutiaPoint

Public Attributes

- bool **has_mru**
- int **mru**
- bool **has_mdu**
- int **mdu**

G.87.1 Detailed Description

Representation of an extended feature set minutia data point.

G.87.2 Member Data Documentation

G.87.2.1 mdu

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaPoint::mdu`

minutiae direction uncertainty

G.87.2.2 mru

`int BiometricEvaluation::Feature::AN2K11EFS::MinutiaPoint::mru`

radius of position uncertainty

G.88 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**

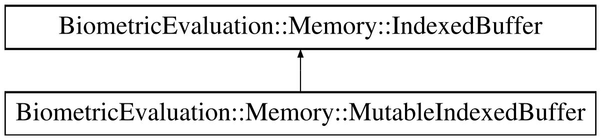
G.88.1 Detailed Description

Representation of a feature point and a set of points.

G.89 BiometricEvaluation::Memory::MutableIndexedBuffer Class Reference

`#include <be_memory_mutableindexedbuffer.h>`

Inheritance diagram for BiometricEvaluation::Memory::MutableIndexedBuffer:


BiometricEvaluation::Memory::IndexedBuffer
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

G.89.1 Detailed Description

Mutable version of an **IndexedBuffer** (p. 391).

G.89.2 Constructor & Destructor Documentation

G.89.2.1 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.

G.89.2.2 MutableIndexedBuffer() [2/3]

```
BiometricEvaluation::Memory::MutableIndexedBuffer::MutableIndexedBuffer (
             uint8Array & aa )
```

Wrap an existing uint8Array.

Parameters

aa	uint8Array to wrap.

G.89.2.3 MutableIndexedBuffer() [3/3]

```
BiometricEvaluation::Memory::MutableIndexedBuffer::MutableIndexedBuffer (
            const MutableIndexedBuffer & copy ) [default]
```

Copy constructor (default).

G.89.2.4 ∼MutableIndexedBuffer()

```
virtual BiometricEvaluation::Memory::MutableIndexedBuffer::~MutableIndexedBuffer ( ) [virtual],
[default]
```

Destructor (default).

G.89.3 Member Function Documentation

G.89.3.1 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. 393).

G.89.3.2 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. 293)	Not enough room to copy len elements.

Returns

The number of elements copied.

G.89.3.3 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. 293)	Not enough room to copy the elements.

Returns

The number of elements copied (2).

G.89.3.4 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. *293)*	Not enough room to copy the elements.

Returns

The number of elements copied (4).

G.89.3.5 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. *293)*	Not enough room to copy the elements.

Returns

The number of elements copied (2).

G.89.3.6 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. *293)*	Not enough room to copy the elements.

Returns

The number of elements copied (4).

G.89.3.7 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. 293)*	Not enough room to copy the elements.

Returns

The number of elements copied (8).

G.89.3.8 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. 293)*	Not enough room to copy the element.

Returns

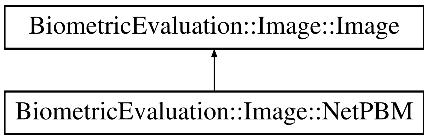
The number of elements copied (1).

G.90 BiometricEvaluation::Image::NetPBM Class Reference

A NetPBM-encoded image.

`#include <be_image_netpbm.h>`

Inheritance diagram for BiometricEvaluation::Image::NetPBM:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::NetPBM

Public Types

- enum **Kind** {
  ASCIIPortableBitmap = 1, **ASCIIPortableGraymap** = 2, **ASCIIPortablePixmap** = 3, **Binary**↩
  PortableBitmap = 4,
  BinaryPortableGraymap = 5, **BinaryPortablePixmap** = 6 }

Public Member Functions

- **NetPBM** (const uint8_t ∗data, const uint64_t size)
- **NetPBM** (const **Memory::uint8Array** &data)
- **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.

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 A↩
  SCIIBufSize, uint32_t width, uint32_t height, uint8_t depth, uint32_t maxColor)

  Convert an ASCII pixel map buffer into a binary pixel map buffer.
- static **Memory::uint8Array BinaryBitmapTo8Bit** (const uint8_t ∗bitmap, uint64_t bitmapSize, uint32↩
  _t width, uint32_t height)

  Convert an binary bitmap (1-bit depth) buffer into an 8-bit depth buffer.

Additional Inherited Members

G.90.1 Detailed Description

A NetPBM-encoded image.

Note

While a **NetPBM** (p. 446) 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. 446) format.

G.90.2 Member Function Documentation

G.90.2.1 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. 542) of bitmap.
width	Width of image in bitmap.
height	Height of image in bitmap.

Returns

8-bit depth representation of bitmap

Exceptions

out_of_range	**Error** (p. 106) extracting a value from the bitmap.

G.90.2.2 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. 542) of ASCIIBuf.
width	Width of image in pixel map.
height	Height of image in pixel map.
depth	Depth of image in pixel map.
maxColor	Maximum color value per pixel. Intensities will be scaled based on this value.

Returns

Binary pixel map representation of the ASCII pixel map in the same depth as the original.

Exceptions

out_of_range	**Error** (p. 106) extracting a value from the pixel map.
Error::ParameterError *(p. 470)*	Invalid value for depth, must be a multiple of 8.

G.90.2.3 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. 542) of bitmap.
width	Width of image in bitmap.
height	Height of image in bitmap.

Returns

8-bit depth representation of bitmap

Exceptions

out_of_range	**Error** (p. 106) extracting a value from the bitmap.

G.90.2.4 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.

Parameters

dataSize	**Size** (p. 542) of data.
offset	Current starting position within data.
sizeOfValue	In the event that the values in data are not space-separated, return a value when it reaches sizeOfValue length. 0 assumes space-separated.

Returns

Next value from data.

G.90.2.5 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. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	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. 357).

G.90.2.6 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. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 470)*	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. 358).

G.90.2.7 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.

G.90.2.8 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. 542) 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.

G.90.2.9 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. 542) 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.

G.91 BiometricEvaluation::Feature::AN2K11EFS::NoFeaturesPresent 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**

G.91.1 Detailed Description

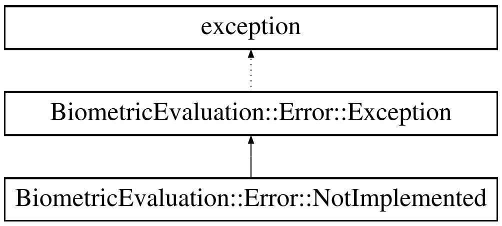
A set of flags indicating "No features present" indicators contained within the extended feature set.

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.

G.92 BiometricEvaluation::Error::NotImplemented Class Reference

A **NotImplemented** (p. 452) 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:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::NotImplemented

Public Member Functions

- **NotImplemented** ()
- **NotImplemented** (const std::string &info)

G.92.1 Detailed Description

A **NotImplemented** (p. 452) object is thrown when the underlying implementation of this interface has not or could not be created.

G.92.2 Constructor & Destructor Documentation

G.92.2.1 NotImplemented() `[1/2]`

```
BiometricEvaluation::Error::NotImplemented::NotImplemented ( )
```

Construct a **NotImplemented** (p. 452) object with the default information string.

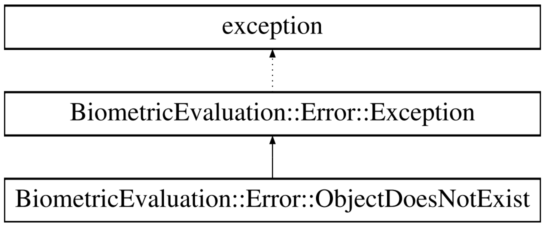
G.92.2.2 NotImplemented() `[2/2]`

```
BiometricEvaluation::Error::NotImplemented::NotImplemented (
            const std::string & info )
```

Construct a **NotImplemented** (p. 452) object with an information string appended to the default information string.

G.93 BiometricEvaluation::Error::ObjectDoesNotExist Class Reference

The named object does not exist.
`#include <be_error_exception.h>`
Inheritance diagram for BiometricEvaluation::Error::ObjectDoesNotExist:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ObjectDoesNotExist

Public Member Functions

- **ObjectDoesNotExist** ()
- **ObjectDoesNotExist** (const std::string &info)

G.93.1 Detailed Description

The named object does not exist.

G.93.2 Constructor & Destructor Documentation

G.93.2.1 ObjectDoesNotExist() `[1/2]`

```
BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( )
```

Construct a **ObjectDoesNotExist** (p. 453) object with the default information string.

G.93.2.2 ObjectDoesNotExist() `[2/2]`

```
BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist (
          const std::string & info )
```

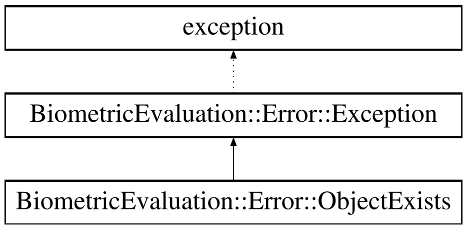
Construct a **ObjectDoesNotExist** (p. 453) object with an information string appended to the default information string.

G.94 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:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ObjectExists

Public Member Functions

- **ObjectExists** ()
- **ObjectExists** (const std::string &info)

G.94.1 Detailed Description

The named object exists and will not be replaced.

G.94.2 Constructor & Destructor Documentation

G.94.2.1 ObjectExists() [1/2]

```
BiometricEvaluation::Error::ObjectExists::ObjectExists ( )
```

Construct a **ObjectExists** (p. 454) object with the default information string.

G.94.2.2 ObjectExists() [2/2]

```
BiometricEvaluation::Error::ObjectExists::ObjectExists (
                const std::string & info )
```

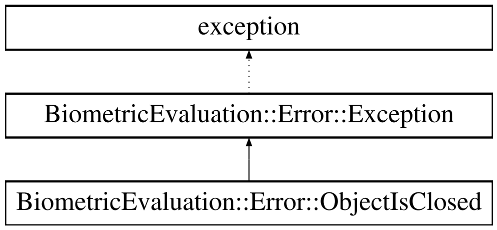
Construct a **ObjectExists** (p. 454) object with an information string appended to the default information string.

G.95 BiometricEvaluation::Error::ObjectIsClosed Class Reference

The object is closed.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectIsClosed:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ObjectIsClosed

Public Member Functions

- **ObjectIsClosed** ()
- **ObjectIsClosed** (const std::string &info)

G.95.1 Detailed Description

The object is closed.

G.95.2 Constructor & Destructor Documentation

G.95.2.1 ObjectIsClosed() [1/2]

```
BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ( )
```

Construct a **ObjectIsClosed** (p. 455) object with the default information string.

G.95.2.2 ObjectIsClosed() [2/2]

```
BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed (
                const std::string & info )
```

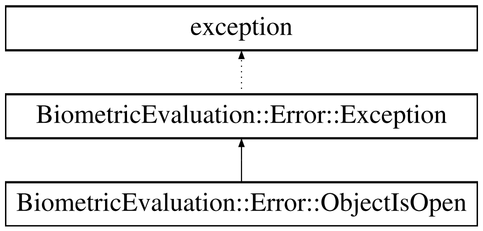
Construct a **ObjectIsClosed** (p. 455) object with an information string appended to the default information string.

G.96 BiometricEvaluation::Error::ObjectIsOpen Class Reference

The object is already opened.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectIsOpen:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ObjectIsOpen

Public Member Functions

- **ObjectIsOpen** ()
- **ObjectIsOpen** (const std::string &info)

G.96.1 Detailed Description

The object is already opened.

G.96.2 Constructor & Destructor Documentation

G.96.2.1 ObjectIsOpen() `[1/2]`

```
BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ( )
```

Construct a **ObjectIsOpen** (p. 456) object with the default information string.

G.96.2.2 ObjectIsOpen() `[2/2]`

```
BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen (
                const std::string & info )
```

Construct a **ObjectIsOpen** (p. 456) object with an information string appended to the default information string.

G.97 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 >**

G.97.1 Detailed Description

template<class Key, class T>
class BiometricEvaluation::Memory::OrderedMap< Key, T >

A map where insertion order is preserved and elements are unique.

G.97.2 Constructor & Destructor Documentation

G.97.2.1 OrderedMap()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >:: OrderedMap ( )
```

Constructor.

G.97.2.2 ∼OrderedMap()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMap< Key, T >::∼ OrderedMap ( )
```

Destructor

G.97.3 Member Function Documentation

G.97.3.1 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.

G.97.3.2 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.

G.97.3.3 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.

G.97.3.4 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.

G.97.3.5 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.

G.97.3.6 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.

G.97.3.7 erase() [1/2]

```
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()).

G.97.3.8 erase() [2/2]

```
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.

G.97.3.9 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).

G.97.3.10 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.

G.97.3.11 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).

G.97.3.12 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.

G.97.3.13 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()).

G.97.3.14 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.

G.98 BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > Class Template Reference

```
#include <be_memory_orderedmap.h>
```

Public Types

- using **iterator_category** = std::bidirectional_iterator_tag
- 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 dummy)
- **OrderedMapConstIterator** & **operator--** ()
- **OrderedMapConstIterator operator--** (int dummy)
- bool **operator==** (const **OrderedMapConstIterator** &rhs) const

  Test for iterator equality.
- bool **operator!=** (const **OrderedMapConstIterator** &rhs) const

  Test for iterator equality.

Friends

- class **OrderedMap< Key, T >**

G.98.1 Detailed Description

template<class Key, class T>
class BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >

Const Iterator for OrderedMaps.

G.98.2 Member Typedef Documentation

G.98.2.1 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

G.98.2.2 iterator_category

```
template<class Key, class T>
using BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >:: iterator_category =
std::bidirectional_iterator_tag
```

Type of iterator

G.98.2.3 pointer

```
template<class Key, class T>
using BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >:: pointer = const value←
_type*
```

Pointer to the type iterated over

G.98.2.4 reference

```
template<class Key, class T>
using BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >:: reference = const value←
_type&
```

Reference to the type iterated over

G.98.2.5 value_type

```
template<class Key, class T>
using BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >:: value_type = std::pair<Key,
T>
```

Type when dereferencing iterators

G.98.3 Constructor & Destructor Documentation

G.98.3.1 OrderedMapConstIterator() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >:: OrderedMapConstIterator ( )
```

Constructor

G.98.3.2 OrderedMapConstIterator() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >:: OrderedMapConstIterator (
            const OrderedMapIterator< Key, T > & iterator )
```

Iterator to ConstIterator converter

G.98.3.3 ∼OrderedMapConstIterator()

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T >::∼ OrderedMapConstIterator (
)
```

Destructor

G.98.4 Member Function Documentation

G.98.4.1 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.

G.98.4.2 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.

G.98.4.3 operator++() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory↩
::OrderedMapConstIterator< Key, T >::operator++ ( )
```

Move to the next pair

G.98.4.4 operator++() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > BiometricEvaluation::Memory↩
::OrderedMapConstIterator< Key, T >::operator++ (
            int dummy )
```

Move to the next pair

G.98.4.5 operator--() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > & BiometricEvaluation::Memory↩
::OrderedMapConstIterator< Key, T >::operator-- ( )
```

Move to the previous pair.

G.98.4.6 operator--() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapConstIterator< Key, T > BiometricEvaluation::Memory←
::OrderedMapConstIterator< Key, T >::operator-- (
            int dummy )
```

Move to the previous pair.

G.98.4.7 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.

G.98.4.8 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.

G.99 BiometricEvaluation::Memory::OrderedMapIterator< Key, T > Class Template Reference

`#include <be_memory_orderedmap.h>`

Public Types

- using **iterator_category** = std::bidirectional_iterator_tag
- 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 dummy)
- **OrderedMapIterator** & **operator--** ()
- **OrderedMapIterator operator--** (int dummy)
- bool **operator==** (const **OrderedMapIterator** &rhs) const

  Test for iterator equality.
- bool **operator!=** (const **OrderedMapIterator** &rhs) const

  Test for iterator equality.

Friends

- class **OrderedMap< Key, T >**
- class **OrderedMapConstIterator< Key, T >**

G.99.1 Detailed Description

template<class Key, class T>
class BiometricEvaluation::Memory::OrderedMapIterator< Key, T >

Iterator for OrderedMaps.

G.99.2 Member Typedef Documentation

G.99.2.1 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

G.99.2.2 iterator_category

```
template<class Key, class T>
using BiometricEvaluation::Memory::OrderedMapIterator< Key, T >:: iterator_category = std↩
::bidirectional_iterator_tag
```

Type of iterator

G.99.2.3 pointer

```
template<class Key, class T>
using BiometricEvaluation::Memory::OrderedMapIterator< Key, T >:: pointer = value_type*
```

Pointer to the type iterated over

G.99.2.4 reference

```
template<class Key, class T>
using BiometricEvaluation::Memory::OrderedMapIterator< Key, T >:: reference = value_type&
```

Reference to the type iterated over

G.99.2.5 value_type

template<class Key, class T>
using **BiometricEvaluation::Memory::OrderedMapIterator**< Key, T >:: **value_type** = std::pair<Key, T>

Type when dereferencing iterators

G.99.3 Constructor & Destructor Documentation

G.99.3.1 OrderedMapIterator()

template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T >:: **OrderedMapIterator** ( )

Constructor

G.99.3.2 ∼OrderedMapIterator()

template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T >::∼ **OrderedMapIterator** ( )

Destructor

G.99.4 Member Function Documentation

G.99.4.1 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.

G.99.4.2 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.

G.99.4.3 operator++() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::↩
OrderedMapIterator< Key, T >::operator++ ( )
```

Move to the next pair

G.99.4.4 operator++() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T > BiometricEvaluation::Memory::Ordered↩
MapIterator< Key, T >::operator++ (
            int dummy )
```

Move to the next pair

G.99.4.5 operator--() [1/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T > & BiometricEvaluation::Memory::↩
OrderedMapIterator< Key, T >::operator-- ( )
```

Move to the previous pair.

G.99.4.6 operator--() [2/2]

```
template<class Key , class T >
BiometricEvaluation::Memory::OrderedMapIterator< Key, T > BiometricEvaluation::Memory::Ordered↩
MapIterator< Key, T >::operator-- (
            int dummy )
```

Move to the previous pair.

G.99.4.7 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.

G.99.4.8 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.

G.100 BiometricEvaluation::Feature::AN2K11EFS::Orientation Struct Reference

Representation of orientation (deviation from upright) and its uncertainty.

`#include <be_feature_an2k11efs.h>`

Public Attributes

- bool **is_default**
- int **eod**
- bool **has_euc**
- int **euc**

Static Public Attributes

- static const int **EODDefault** = 0
- static const int **EUCDefault** = 15

G.100.1 Detailed Description

Representation of orientation (deviation from upright) and its uncertainty.

G.100.2 Member Data Documentation

G.100.2.1 eod

`int BiometricEvaluation::Feature::AN2K11EFS::Orientation::eod`

Direction

G.100.2.2 EODDefault

`const int BiometricEvaluation::Feature::AN2K11EFS::Orientation::EODDefault = 0 [static]`

ANSI/NIST default direction

G.100.2.3 euc

`int BiometricEvaluation::Feature::AN2K11EFS::Orientation::euc`

Uncertainty

G.100.2.4 EUCDefault

`const int BiometricEvaluation::Feature::AN2K11EFS::Orientation::EUCDefault = 15 [static]`

ANSI/NIST default uncertainty

G.100.2.5 is_default

`bool BiometricEvaluation::Feature::AN2K11EFS::Orientation::is_default`

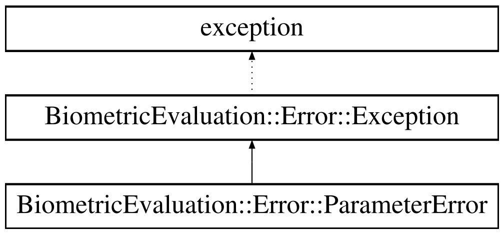
Whether the values are the defaults

G.101 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:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ParameterError

Public Member Functions

- **ParameterError** ()
- **ParameterError** (const std::string &info)

G.101.1 Detailed Description

An invalid parameter was passed to a constructor or method.

G.101.2 Constructor & Destructor Documentation

G.101.2.1 ParameterError() `[1/2]`

`BiometricEvaluation::Error::ParameterError::ParameterError ( )`

Construct a **ParameterError** (p. 470) object with the default information string.

G.101.2.2 ParameterError() `[2/2]`

```
BiometricEvaluation::Error::ParameterError::ParameterError (
            const std::string & info )
```

Construct a **ParameterError** (p. 470) object with an information string appended to the default information string.

G.102 BiometricEvaluation::Feature::AN2K7Minutiae::PatternClassification Class Reference

Pattern classification codes.

`#include <be_feature_an2k7minutiae.h>`

Classes

- struct **Entry**

Public Types

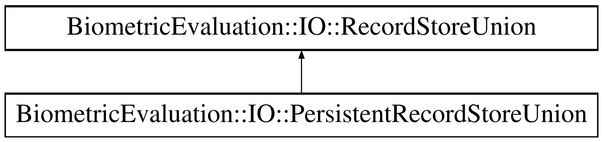
- using **Entry** = struct **Entry**

G.102.1 Detailed Description

Pattern classification codes.

G.103 BiometricEvaluation::IO::PersistentRecordStoreUnion Class Reference

Inheritance diagram for BiometricEvaluation::IO::PersistentRecordStoreUnion:


BiometricEvaluation::IO::RecordStoreUnion
BiometricEvaluation::IO::PersistentRecordStoreUnion

Public Member Functions

- **PersistentRecordStoreUnion** (const std::string &path)

  Open an existing ***PersistentRecordStoreUnion*** *(p. 471).*
- **PersistentRecordStoreUnion** (const std::string &path, const std::map< const std::string, const std↩ ::string > &recordStores)

  Create a new ***PersistentRecordStoreUnion*** *(p. 471).*
- **PersistentRecordStoreUnion** (const std::string &path, std::initializer_list< std::pair< const std::string, const std::string >> &recordStores)

  Create a new ***PersistentRecordStoreUnion*** *(p. 471).*
- ∼**PersistentRecordStoreUnion** ()=default

Additional Inherited Members

G.103.1 Constructor & Destructor Documentation

G.103.1.1 PersistentRecordStoreUnion() [1/3]

```
BiometricEvaluation::IO::PersistentRecordStoreUnion::PersistentRecordStoreUnion (
            const std::string & path )
```

Open an existing **PersistentRecordStoreUnion** (p. 471).

Parameters

path	Path at which **RecordStoreUnion** (p. 521) was persisted.

G.103.1.2 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. 471).

Parameters

path	Path at which **RecordStoreUnion** (p. 521) will be persisted.
recordStores	Initial RecordStores members of the union.

G.103.1.3 PersistentRecordStoreUnion() [3/3]

```
BiometricEvaluation::IO::PersistentRecordStoreUnion::PersistentRecordStoreUnion (
            const std::string & path,
            std::initializer_list< std::pair< const std::string, const std::string >> & record↩
Stores )
```

Create a new **PersistentRecordStoreUnion** (p. 471).

Parameters

path	Path at which **RecordStoreUnion** (p. 521) will be persisted.
mode	Mode in which to open RecordStores in the union.
recordStores	Initial RecordStores members of the union.

G.103.1.4 ∼PersistentRecordStoreUnion()

```
BiometricEvaluation::IO::PersistentRecordStoreUnion::∼PersistentRecordStoreUnion ( )  [default]
```

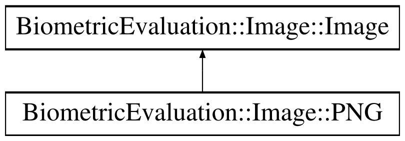
Destructor

G.104 BiometricEvaluation::Image::PNG Class Reference

A PNG-encoded image.

```
#include <be_image_png.h>
```

Inheritance diagram for BiometricEvaluation::Image::PNG:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::PNG

Public Member Functions

- **PNG** (const uint8_t ∗data, const uint64_t size)
- **PNG** (const **Memory::uint8Array** &data)
- **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.

Static Public Member Functions

- static bool **isPNG** (const uint8_t ∗data, uint64_t size)

Additional Inherited Members

G.104.1 Detailed Description

A PNG-encoded image.

G.104.2 Member Function Documentation

G.104.2.1 getRawData()

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

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 357).

G.104.2.2 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. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 470)*	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. 358).

G.104.2.3 isPNG()

```
static bool BiometricEvaluation::Image::PNG::isPNG (
            const uint8_t * data,
            uint64_t size )  [static]
```

Whether or not data is a **PNG** (p. 472) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a **PNG** (p. 472) image, false otherwise

G.105 BiometricEvaluation::Feature::Sort::Polar Class Reference

Sort (p. 111) by increasing distance from center and angle (theta).

`#include <be_feature_sort.h>`

Public Member Functions

- **Polar** (const **BiometricEvaluation::Image::Coordinate** &center)

  Polar *(p. 474) 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.

G.105.1 Detailed Description

Sort (p. 111) by increasing distance from center and angle (theta).

G.105.2 Constructor & Destructor Documentation

G.105.2.1 Polar()

```
BiometricEvaluation::Feature::Sort::Polar::Polar (
              const  BiometricEvaluation::Image::Coordinate & center )
```

Polar (p. 474) constructor.

Parameters

center	Coordinate to use for center of image.

centerOfMinutiaeMass centerOfImage

G.105.3 Member Function Documentation

G.105.3.1 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.

G.105.3.2 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. 560)*	No minutia.

G.105.3.3 operator()()

```
bool BiometricEvaluation::Feature::Sort::Polar::operator() (
            const  BiometricEvaluation::Feature::MinutiaPoint & lhs,
            const  BiometricEvaluation::Feature::MinutiaPoint & rhs ) const
```

MinutiaPoint (p. 440) polar ascending comparator.

G.106 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**

G.106.1 Detailed Description

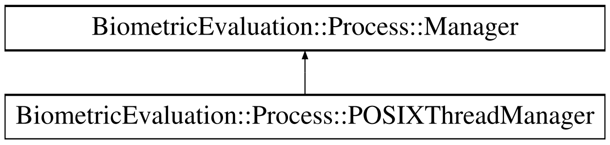
Representation of pose angle and uncertainty.

G.107 BiometricEvaluation::Process::POSIXThreadManager Class Reference

Manager (p. 426) implementation that starts Workers in POSIX threads.

`#include <be_process_posixthreadmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::POSIXThreadManager:


BiometricEvaluation::Process::Manager
BiometricEvaluation::Process::POSIXThreadManager

Public Member Functions

- **POSIXThreadManager** ()
- std::shared_ptr< **WorkerController** > **addWorker** (std::shared_ptr< **Worker** > worker)

  *Adds a **Worker** (p. 588) to be managed by this **Manager** (p. 426).*
- void **startWorkers** (bool wait=true, bool communicate=false)

  *Begin **Worker** (p. 588)'s work.*
- void **startWorker** (std::shared_ptr< **WorkerController** > worker, bool wait=true, bool communicate=false)

  *Start a **Worker** (p. 588).*
- void **stopWorker** (std::shared_ptr< **WorkerController** > workerController)

  *Ask **Worker** (p. 588) to exit.*
- void **waitForWorkerExit** ()

  Block until all Workers have exited.
- ∼**POSIXThreadManager** ()

  ∼POSIXThreadManager destructor.

Additional Inherited Members

G.107.1 Detailed Description

Manager (p. 426) implementation that starts Workers in POSIX threads.

G.107.2 Constructor & Destructor Documentation

G.107.2.1 POSIXThreadManager()

```
BiometricEvaluation::Process::POSIXThreadManager::POSIXThreadManager ( )
```

POSIXThreadManager (p. 476) constructor.

G.107.3 Member Function Documentation

G.107.3.1 addWorker()

```
std::shared_ptr< WorkerController> BiometricEvaluation::Process::POSIXThreadManager::addWorker
(
              std::shared_ptr< Worker > worker ) [virtual]
```

Adds a **Worker** (p. 588) to be managed by this **Manager** (p. 426).

Parameters

worker	A **Worker** (p. 588) instance to run.

Returns

shared_ptr to worker.

Implements **BiometricEvaluation::Process::Manager** (p. 427).

G.107.3.2 startWorker()

```
void BiometricEvaluation::Process::POSIXThreadManager::startWorker (
             std::shared_ptr< WorkerController > worker,
             bool wait = true,
             bool communicate = false )  [virtual]
```

Start a **Worker** (p. 588).

Parameters

worker	Pointer to a **WorkerController** (p. 595) that is being managed by this **Manager** (p. 426) instance.
wait	Whether or not to wait for this **Worker** (p. 588) to exit before returning control to the caller.
communicate	Whether or not to enable communication among the Workers and Managers.

Exceptions

Error::ObjectExists *(p. 454)*	worker is already working.
Error::StrategyError *(p. 560)*	worker is not managed by this **Manager** (p. 426) instance.

Implements **BiometricEvaluation::Process::Manager** (p. 429).

G.107.3.3 startWorkers()

```
void BiometricEvaluation::Process::POSIXThreadManager::startWorkers (
             bool wait = true,
             bool communicate = false )  [virtual]
```

Begin **Worker** (p. 588)'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. 454)*	At least one **Worker** (p. 588) is already working.
Error::StrategyError *(p. 560)*	Problem starting the Workers.

Implements **BiometricEvaluation::Process::Manager** (p. 430).

G.107.3.4 stopWorker()

```
void BiometricEvaluation::Process::POSIXThreadManager::stopWorker (
            std::shared_ptr< WorkerController > workerController )  [virtual]
```

Ask **Worker** (p. 588) to exit.

Parameters

workerController	Pointer to the **WorkerController** (p. 595) that should be stopped.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	worker is not working.
Error::StrategyError *(p. 560)*	Problem sending the signal.

Implements **BiometricEvaluation::Process::Manager** (p. 430).

G.107.3.5 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.

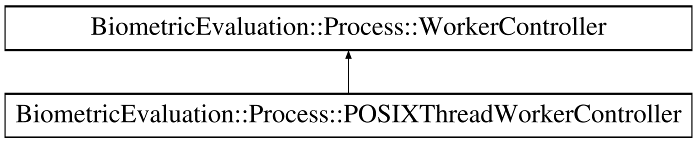
Implements **BiometricEvaluation::Process::Manager** (p. 431).

G.108 BiometricEvaluation::Process::POSIXThreadWorkerController Class Reference

Decorated **Worker** (p. 588) returned from a **Process::POSIXThreadManager** (p. 476).

`#include <be_process_posixthreadmanager.h>`

Inheritance diagram for BiometricEvaluation::Process::POSIXThreadWorkerController:


BiometricEvaluation::Process::WorkerController
BiometricEvaluation::Process::POSIXThreadWorkerController

Public Member Functions

- void **reset** ()

  *Reuse the **Worker** (p. 588).*
- bool **isWorking** () const

  *Obtain whether or not **Worker** (p. 588) is working.*
- bool **everWorked** () const

  *Obtain whether or not this **Worker** (p. 588) has ever worked.*
- ∼**POSIXThreadWorkerController** ()

  ***POSIXThreadWorkerController** (p. 479) destructor.*

Friends

- class **POSIXThreadManager**

Additional Inherited Members

G.108.1 Detailed Description

Decorated **Worker** (p. 588) returned from a **Process::POSIXThreadManager** (p. 476).

G.108.2 Member Function Documentation

G.108.2.1 everWorked()

```
bool BiometricEvaluation::Process::POSIXThreadWorkerController::everWorked ( ) const  [virtual]
```

Obtain whether or not this **Worker** (p. 588) has ever worked.

Returns

true the **Worker** (p. 588) has ever or is currently working, false otherwise.

Note

reset() (p. 480) will change the result of this method.

Implements **BiometricEvaluation::Process::WorkerController** (p. 596).

G.108.2.2 isWorking()

```
bool BiometricEvaluation::Process::POSIXThreadWorkerController::isWorking ( ) const  [virtual]
```

Obtain whether or not **Worker** (p. 588) is working.

Returns

Whether or not the **Worker** (p. 588) is working.

Implements **BiometricEvaluation::Process::WorkerController** (p. 597).

G.108.2.3 reset()

```
void BiometricEvaluation::Process::POSIXThreadWorkerController::reset ( )  [virtual]
```

Reuse the **Worker** (p. 588).

Exceptions

Error::ObjectExists *(p. 454)*	The previously started **Worker** (p. 588) is still running.

Reimplemented from **BiometricEvaluation::Process::WorkerController** (p. 597).

G.109 BiometricEvaluation::View::AN2KViewVariableResolution::Print←PositionCoordinate 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**

G.109.1 Detailed Description

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

G.109.2 Member Data Documentation

G.109.2.1 coordinates

```
Image::CoordinateSet BiometricEvaluation::View::AN2KViewVariableResolution::PrintPositionCoordinate←
::coordinates
```

Two coordinates forming bounding box

G.109.2.2 fingerView

```
Finger::FingerImageCode BiometricEvaluation::View::AN2KViewVariableResolution::PrintPosition←
Coordinate::fingerView
```

Full finger view being bounded

G.109.2.3 segment

```
Finger::FingerImageCode BiometricEvaluation::View::AN2KViewVariableResolution::PrintPosition←
Coordinate::segment
```

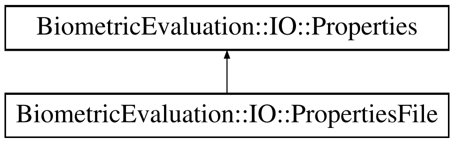
Segment within full finger view bound

G.110 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:


BiometricEvaluation::IO::Properties
BiometricEvaluation::IO::PropertiesFile

Public Member Functions

- **Properties** (**IO::Mode** mode= **IO::Mode::ReadWrite**, const std::map< std::string, std::string > &defaults={})

  *Construct a new **Properties** (p. 482) 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. 482) 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. 482) 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.

G.110.1 Detailed Description

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

G.110.2 Constructor & Destructor Documentation

G.110.2.1 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. 482) object.

Parameters

in	*mode*	The read/write mode of the object.
in	*defaults*	Default property/value pairs to insert.

G.110.2.2 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. 482) object from the contents of a buffer.
The format of the buffer can be seen in **PropertiesFile** (p. 489).

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. 560)*	A line in the properties file is malformed.

G.110.2.3 ∼Properties()

```
virtual BiometricEvaluation::IO::Properties::∼Properties ( )  [virtual]
```

Destructor

G.110.3 Member Function Documentation

G.110.3.1 getMode()

```
BiometricEvaluation::IO::Mode BiometricEvaluation::IO::Properties::getMode ( ) const  [protected]
```

Obtain the mode of the **Properties** (p. 482) object.

Returns

Mode (**Mode::ReadOnly** (p. 125) or **Mode::ReadWrite** (p. 125))

G.110.3.2 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. 453)*	The named property does not exist.

G.110.3.3 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. 453)*	The named property does not exist.
Error::ConversionError *(p. 282)*	The property value cannot be converted, due to non-numeric characters in the string, or the value is the empty string.

G.110.3.4 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. 453)*	The named property does not exist.
Error::ConversionError *(p. 282)*	The property value cannot be converted, due to non-numeric characters in the string, or the value is the empty string.

G.110.3.5 getPropertyKeys()

```
std::vector<std::string> BiometricEvaluation::IO::Properties::getPropertyKeys ( ) const
```

Retrieve a set of all property keys.

Returns

A vector of property key strings.

G.110.3.6 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. 560)*	A line of the buffer is malformed.

G.110.3.7 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. *560*)	A line of the buffer is malformed.

G.110.3.8 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. *453*)	The named property does not exist.
Error::StrategyError (p. *560*)	The **Properties** (p. 482) object is read-only.

G.110.3.9 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. 560)*	The **Properties** (p. 482) object is read-only.

G.110.3.10 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. 560)*	The **Properties** (p. 482) object is read-only.

G.110.3.11 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. *560*)	The **Properties** (p. 482) object is read-only.

G.110.3.12 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

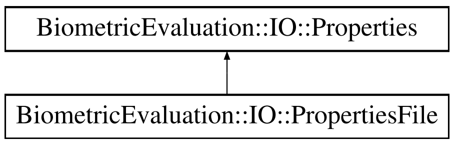
Error::StrategyError (p. *560*)	The **Properties** (p. 482) object is read-only.

G.111 BiometricEvaluation::IO::PropertiesFile Class Reference

A **Properties** (p. 482) object persisted in an file on disk.

`#include <be_io_propertiesfile.h>`

Inheritance diagram for BiometricEvaluation::IO::PropertiesFile:


BiometricEvaluation::IO::Properties
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. 482) 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. 482), which means changing the name of the underlying file that stores the properties.*

- ∼**PropertiesFile** ()
- **PropertiesFile** (const **PropertiesFile** &other)=delete

  Copy constructor (disabled).
- **PropertiesFile** & **operator=** (const **PropertiesFile** &other)=delete

  Assignment operator (disabled).

Additional Inherited Members

G.111.1 Detailed Description

A **Properties** (p. 482) object persisted in an file on disk.
An example file might look like this:

```
*     Name = John Smith
*     Age = 32
*     Favorite Hex Number = 0xffff
*
```

For property keys and values, leading and trailing whitespace is removed, therefore the call

```
props->setProperty("  My property   ", "   A Value  ");
```

results in an entry in the property file as

```
*     My property = A value
*
```

Therefore, the property names "Foo", " Foo", "Foo " are equivalent.

G.111.2 Constructor & Destructor Documentation

G.111.2.1 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. 482) object from an existing or to be created properties file. The constructor will create the file when it does not exist.

Parameters

in	*pathname*	The path to the file to store the properties.
in	*mode*	The read/write mode of the object.
in	*defaults*	Default property/value pairs to insert.

Exceptions

Error::StrategyError *(p. 560)*	A line in the properties file is malformed.
Error::FileError *(p. 312)*	An error occurred when using the underlying storage system.

G.111.2.2 ∼PropertiesFile()

```
BiometricEvaluation::IO::PropertiesFile::∼PropertiesFile ( )
```

Destructor

G.111.2.3 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. 489) object to copy.

G.111.3 Member Function Documentation

G.111.3.1 changeName()

```
void BiometricEvaluation::IO::PropertiesFile::changeName (
             const std::string & pathname )
```

Change the name of the **Properties** (p. 482), 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. 482) file.

Exceptions

Error::StrategyError *(p. 560)*	The object is read-only.
Error::ObjectExists *(p. 454)*	A file at pathname already exists.

G.111.3.2 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. 489) object to assign;

Returns

This **PropertiesFile** (p. 489) object, now containing the contents of other.

G.111.3.3 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. 312)*	An error occurred when using the underlying storage system.
Error::StrategyError *(p. 560)*	The object was constructed with nullptr as the file name, or is read-only.

G.112 BiometricEvaluation::Feature::Sort::Quality Class Reference

```
#include <be_feature_sort.h>
```

Public Member Functions

- bool **operator()** (const **BiometricEvaluation::Feature::MinutiaPoint** &lhs, const **Biometric↩ Evaluation::Feature::MinutiaPoint** &rhs) const

    MinutiaPoint *(p. 440) quality ascending comparator.*

G.112.1 Detailed Description

Sort (p. 111) by increasing minutiae quality

G.113 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**

G.113.1 Detailed Description

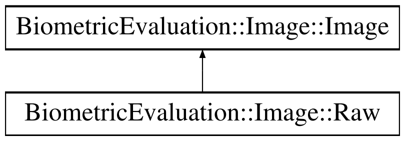
Representation of an iris quality block.

G.114 BiometricEvaluation::Image::Raw Class Reference

An image with no encoding or compression.

`#include <be_image_raw.h>`

Inheritance diagram for BiometricEvaluation::Image::Raw:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::Raw

Public Member Functions

- **Raw** (const uint8_t *data, const uint64_t size, const **Size** dimensions, const uint32_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const bool **hasAlphaChannel**)
- **Raw** (const **BiometricEvaluation::Memory::uint8Array** &data, const **Size** dimensions, const uint32↩_t colorDepth, const uint16_t bitDepth, const **Resolution** resolution, const bool **hasAlphaChannel**)
- **Memory::uint8Array getRawData** () const

  Accessor for the raw image data. The data returned should not be compressed or encoded.
- **Memory::uint8Array getRawGrayscaleData** (uint8_t depth) const

  Accessor for decompressed data in grayscale.

Additional Inherited Members

G.114.1 Detailed Description

An image with no encoding or compression.

G.114.2 Member Function Documentation

G.114.2.1 getRawData()

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

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 357).

G.114.2.2 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. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 470)*	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. 358).

G.115 BiometricEvaluation::MPI::Receiver Class Reference

A class to represent an **MPI** (p. 143) task that receives WorkPackages containers from the **Distributor** (p. 303).

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

Public Member Functions

- **Receiver** (const std::string &propertiesFileName, const std::shared_ptr< **BiometricEvaluation::M↩ PI::WorkPackageProcessor** > &workPackageProcessor)

  Construct a new work package receiver.
- void **start** ()

  Start the receiving task.

G.115.1 Detailed Description

A class to represent an **MPI** (p. 143) task that receives WorkPackages containers from the **Distributor** (p. 303).

A receiver object depends on a set of properties contained in a file. The properties specify **MPI** (p. 143) 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. 495) is called. The receiver will start workers only when the distributor indicates that it has started successfully. Otherwise, the **Receiver** (p. 494) 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. 143) task created by the **MPI** (p. 143) runtime, and the child process created by **Receiver** (p. 494).

See also

IO::Properties (p. 482)
IO::Logsheet (p. 416)
MPI::Distributor (p. 303)
Process::Worker (p. 588)

G.115.2 Constructor & Destructor Documentation

G.115.2.1 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. 307)*	An error occurred when constructing this object.

G.115.3 Member Function Documentation

G.115.3.1 start()

```
void BiometricEvaluation::MPI::Receiver::start ( )
```

Start the receiving task.

Upon starting, the **Receiver** (p. 494) object will begin communicating with the **Distributor** (p. 303) using **MPI** (p. 143) messages. This **Receiver** (p. 494) object will send a status message back to the **Distributor** (p. 303) indicating success or failure to initialize. Success includes the startup of at least one worker process.

G.116 BiometricEvaluation::IO::RecordStore::Record Struct Reference

Public Member Functions

- **Record** ()
- **Record** (const std::string &key, const **Memory::uint8Array** &data)

  *Create a **Record** (p. 496) from the key and data.*

Public Attributes

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

G.116.1 Constructor & Destructor Documentation

G.116.1.1 Record() [1/2]

```
BiometricEvaluation::IO::RecordStore::Record::Record ( )
```

Default constructor.

G.116.1.2 Record() [2/2]

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

Create a **Record** (p. 496) from the key and data.

Parameters

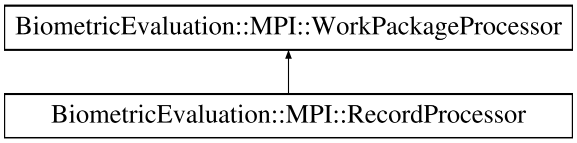
in	*key*	The record's key.
in	*data*	The record's data (value).

G.117 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. 600).

```
#include <be_mpi_recordprocessor.h>
```

Inheritance diagram for BiometricEvaluation::MPI::RecordProcessor:


BiometricEvaluation::MPI::WorkPackageProcessor
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. 148) the data contents of the work package. This method is part of the worker personality.*

Protected Member Functions

- std::shared_ptr< **MPI::RecordStoreResources** > **getResources** ()

G.117.1 Detailed Description

An implementation of a work package processor that will extract record store keys, and optionally, values, from a **WorkPackage** (p. 600).

Subclasses of this abstract class must implement the method to process the records associated with the keys.

G.117.2 Constructor & Destructor Documentation

G.117.2.1 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. 307)*	An error occurred, usually due to missing or incorrect properties.

G.117.3 Member Function Documentation

G.117.3.1 newProcessor()

```
virtual std::shared_ptr< WorkPackageProcessor> BiometricEvaluation::MPI::RecordProcessor::new←↩
Processor (
            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. 416) 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. 602). If an error occurs during construction, throw a **Error::Exception** (p. 307) with a message to be caught and logged.

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 603).

G.117.3.2 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. 416) that may be used to save messages generated by the object.

Exceptions

Error::Exception *(p. 307)*	An implementation specific error occurred. The exception string will be logged by the **Framework** (p. 115).

Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 603).

G.117.3.3 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. 307)*	An error occurred processing the record: Missing record, input/output error, or memory allocation.

G.117.3.4 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. 307)*	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.

G.117.3.5 processWorkPackage()

```
void BiometricEvaluation::MPI::RecordProcessor::processWorkPackage (
             MPI::WorkPackage & workPackage )  [virtual]
```

Process (p. 148) 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. 307)*	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.

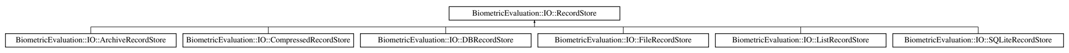
Implements **BiometricEvaluation::MPI::WorkPackageProcessor** (p. 604).

G.118 BiometricEvaluation::IO::RecordStore Class Reference

A class to represent a data storage mechanism.

`#include <be_io_recordstore.h>`

Inheritance diagram for BiometricEvaluation::IO::RecordStore:


BiometricEvaluation::IO::RecordStore
BiometricEvaluation::IO::ArchiveRecordStore
BiometricEvaluation::IO::CompressedRecordStore
BiometricEvaluation::IO::DBRecordStore
BiometricEvaluation::IO::FileRecordStore
BiometricEvaluation::IO::ListRecordStore
BiometricEvaluation::IO::SQLiteRecordStore

Classes

- struct **Record**

Public Types

- enum **Kind** {
  Kind::BerkeleyDB, **Kind::Archive**, **Kind::File**, **Kind::SQLite**,
  Kind::Compressed, **Kind::List**, **Kind::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. 500).*
- 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. *500*)*, returning the key/data pairs.*
- virtual std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**)=0

  Sequence through a ***RecordStore*** (p. *500*)*, returning the key.*
- virtual void **setCursorAtKey** (const std::string &key)=0
- virtual bool **containsKey** (const std::string &key) const

  Determines whether the ***RecordStore*** (p. *500*) *contains an element with the specified key.*
- virtual **iterator** **begin** () noexcept
- virtual **iterator** **end** () noexcept

Static Public Member Functions

- static std::shared_ptr< **RecordStore** > **openRecordStore** (const std::string &pathname, **IO::Mode** mode= **Mode::ReadOnly**)

  Open an existing ***RecordStore*** (p. *500*) *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. *500*) *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)

  Create a new ***RecordStore*** (p. *500*) *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

G.118.1 Detailed Description

A class to represent a data storage mechanism.

A **RecordStore** (p. 500) 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. 513). A key string cannot begin with the space character.

See also

IO::ArchiveRecordStore (p. 222), **IO::DBRecordStore** (p. 293), **IO::FileRecordStore** (p. 321).

G.118.2 Member Enumeration Documentation

G.118.2.1 Kind

```
enum BiometricEvaluation::IO::RecordStore::Kind [strong]
```

Possible types of **RecordStore** (p. 500)

Enumerator

BerkeleyDB	**DBRecordStore** (p. 293)
Archive	**ArchiveRecordStore** (p. 222)
File	**FileRecordStore** (p. 321)
SQLite	**SQLiteRecordStore** (p. 547)
Compressed	**CompressedRecordStore** (p. 260)
List	**ListRecordStore** (p. 409)
Default	”Default” **RecordStore** (p. 500) kind

G.118.3 Member Function Documentation

G.118.3.1 begin()

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

Returns

Iterator to the first record.

G.118.3.2 changeDescription()

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

Change the description of the **RecordStore** (p. 500).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 225), **BiometricEvaluation::IO↩ ::ListRecordStore** (p. 410), **BiometricEvaluation::IO::FileRecordStore** (p. 323), **BiometricEvaluation↩ ::IO::DBRecordStore** (p. 295), **BiometricEvaluation::IO::CompressedRecordStore** (p. 263), and **Biometric↩ Evaluation::IO::SQLiteRecordStore** (p. 548).

G.118.3.3 containsKey()

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

Determines whether the **RecordStore** (p. 500) contains an element with the specified key.

Parameters

key	The key to locate.

Returns

True if the **RecordStore** (p. 500) contains an element with the key, false otherwise.

G.118.3.4 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. 500) 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. 500) to be created.

Returns

An managed pointer to the object representing the created store.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	The **RecordStore** (p. 500) does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

G.118.3.5 end()

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

Returns

Iterator past the last record.

G.118.3.6 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::CompressedRecordStore** (p. 263), **BiometricEvaluation::IO::ArchiveRecordStore** (p. 225), **BiometricEvaluation::IO::FileRecordStore** (p. 324), **BiometricEvaluation::IO::DBRecordStore** (p. 296), **BiometricEvaluation::IO::ListRecordStore** (p. 411), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 549).

G.118.3.7 getCount()

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

Obtain the number of items in the **RecordStore** (p. 500).

Returns

The number of items in the **RecordStore** (p. 500).

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 226), **BiometricEvaluation::IO::ListRecordStore** (p. 411), **BiometricEvaluation::IO::FileRecordStore** (p. 324), **BiometricEvaluation::IO::DBRecordStore** (p. 296), **BiometricEvaluation::IO::CompressedRecordStore** (p. 264), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 549).

G.118.3.8 getDescription()

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

Obtain a textual description of the **RecordStore** (p. 500).

Returns

The **RecordStore** (p. 500)'s description.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 226), **BiometricEvaluation::IO::ListRecordStore** (p. 411), **BiometricEvaluation::IO::FileRecordStore** (p. 324), **BiometricEvaluation::IO::DBRecordStore** (p. 296), **BiometricEvaluation::IO::CompressedRecordStore** (p. 264), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 549).

G.118.3.9 getPathname()

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

Return the path name of the **RecordStore** (p. 500).

Returns

Where in the file system the **RecordStore** (p. 500) is located.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 226), **BiometricEvaluation::IO::ListRecordStore** (p. 411), **BiometricEvaluation::IO::FileRecordStore** (p. 324), **BiometricEvaluation::IO::DBRecordStore** (p. 296), **BiometricEvaluation::IO::CompressedRecordStore** (p. 264), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 550).

G.118.3.10 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. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::ArchiveRecordStore** (p. 226), **BiometricEvaluation::IO::FileRecordStore** (p. 324), **BiometricEvaluation::IO::ListRecordStore** (p. 411), **BiometricEvaluation::IO::DBRecordStore** (p. 296), **BiometricEvaluation::IO::CompressedRecordStore** (p. 264), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 550).

G.118.3.11 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. 454)*	A record with the given key is already present.
Error::StrategyError *(p. 560)*	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

G.118.3.12 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. 454)*	A record with the given key is already present.
Error::StrategyError *(p. 560)*	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::CompressedRecordStore** (p. 265), **BiometricEvaluation::IO::ArchiveRecordStore** (p. 227), **BiometricEvaluation::IO::DBRecordStore** (p. 297), **BiometricEvaluation::IO::FileRecordStore** (p. 325), **BiometricEvaluation::IO::ListRecordStore** (p. 412), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 550).

G.118.3.13 length()

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

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::CompressedRecordStore** (p. 265), **BiometricEvaluation↩ ::IO::ArchiveRecordStore** (p. 227), **BiometricEvaluation::IO::FileRecordStore** (p. 325), **Biometric↩ Evaluation::IO::DBRecordStore** (p. 297), **BiometricEvaluation::IO::ListRecordStore** (p. 412), and **Biometric↩ Evaluation::IO::SQLiteRecordStore** (p. 551).

G.118.3.14 mergeRecordStores()

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

Create a new **RecordStore** (p. 500) that contains the contents of several other RecordStores.

Parameters

in	*mergePathname*	The path name of the new **RecordStore** (p. 500) that will be created.
in	*description*	The text used to describe the new **RecordStore** (p. 500).
in	*kind*	The kind of the new, merged **RecordStore** (p. 500).
in	*pathnames*	Vector of path names to RecordStores to open. These are the RecordStores that will be merged to create the new **RecordStore** (p. 500).

Exceptions

Error::ObjectExists *(p. 454)*	A **RecordStore** (p. 500) at mergePathname already exists.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

G.118.3.15 move()

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

Move the **RecordStore** (p. 500).

The **RecordStore** (p. 500) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::CompressedRecordStore** (p. 266), **BiometricEvaluation::IO::ArchiveRecordStore** (p. 228), **BiometricEvaluation::IO::FileRecordStore** (p. 326), **BiometricEvaluation::IO::ListRecordStore** (p. 413), **BiometricEvaluation::IO::DBRecordStore** (p. 298), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 551).

G.118.3.16 openRecordStore()

```
static std::shared_ptr< RecordStore> BiometricEvaluation::IO::RecordStore::openRecordStore (
            const std::string & pathname,
             IO::Mode mode =  Mode::ReadOnly )  [static]
```

Open an existing **RecordStore** (p. 500) 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. 500) it is.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

in	*pathname*	The path name of the store to be opened.
in	*mode*	The type of access a client of this **RecordStore** (p. 500) has.

Returns

An object representing the existing store.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	The **RecordStore** (p. 500) does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

G.118.3.17 read()

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

Read a complete record from a store.

The AutoArray will be resized to match the size of the data.

Parameters

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

Returns

The record associated with the key.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::CompressedRecordStore** (p. 266), **BiometricEvaluation::IO::ArchiveRecordStore** (p. 229), **BiometricEvaluation::IO::FileRecordStore** (p. 326), **BiometricEvaluation::IO::DBRecordStore** (p. 298), **BiometricEvaluation::IO::ListRecordStore** (p. 413), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 552).

G.118.3.18 remove()

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

Remove a record from the store.

Parameters

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

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::CompressedRecordStore** (p. 267), **BiometricEvaluation::IO::ArchiveRecordStore** (p. 229), **BiometricEvaluation::IO::DBRecordStore** (p. 299), **BiometricEvaluation::IO::FileRecordStore** (p. 327), **BiometricEvaluation::IO::ListRecordStore** (p. 414), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 552).

G.118.3.19 removeRecordStore()

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

Remove a **RecordStore** (p. 500) by deleting all persistant data associated with the store.

Parameters

in	*pathname*	The name of the existing **RecordStore** (p. 500).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record with the given key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

G.118.3.20 replace() [1/2]

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

Replace a complete record in a **RecordStore** (p. 500).

Parameters

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

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

G.118.3.21 replace() [2/2]

```
virtual void BiometricEvaluation::IO::RecordStore::replace (
             const std::string & key,
             const void *const data,
             const uint64_t size )  [virtual]
```

Replace a complete record in a **RecordStore** (p. 500).

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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Reimplemented in **BiometricEvaluation::IO::FileRecordStore** (p. 327), and **BiometricEvaluation::↩ IO::ListRecordStore** (p. 414).

G.118.3.22 sequence()

```
virtual  RecordStore::Record BiometricEvaluation::IO::RecordStore::sequence (
            int cursor =  BE_RECSTORE_SEQ_NEXT )  [pure virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::CompressedRecordStore** (p. 267), **BiometricEvaluation↩ ::IO::ArchiveRecordStore** (p. 230), **BiometricEvaluation::IO::FileRecordStore** (p. 327), **Biometric↩ Evaluation::IO::ListRecordStore** (p. 414), **BiometricEvaluation::IO::DBRecordStore** (p. 299), and **Biometric↩ Evaluation::IO::SQLiteRecordStore** (p. 552).

G.118.3.23 sequenceKey()

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

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::CompressedRecordStore** (p. 268), **BiometricEvaluation::IO::ArchiveRecordStore** (p. 230), **BiometricEvaluation::IO::FileRecordStore** (p. 328), **BiometricEvaluation::IO::ListRecordStore** (p. 415), **BiometricEvaluation::IO::DBRecordStore** (p. 299), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 553).

G.118.3.24 setCursorAtKey()

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

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 500), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 511).

Parameters

`in`	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 511).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::CompressedRecordStore** (p. 268), **BiometricEvaluation::IO::ArchiveRecordStore** (p. 231), **BiometricEvaluation::IO::FileRecordStore** (p. 328), **BiometricEvaluation::IO::ListRecordStore** (p. 415), **BiometricEvaluation::IO::DBRecordStore** (p. 300), and **BiometricEvaluation::IO::SQLiteRecordStore** (p. 553).

G.118.3.25 sync()

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

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implemented in **BiometricEvaluation::IO::FileRecordStore** (p. 329), **BiometricEvaluation::IO::DBRecordStore** (p. 300), **BiometricEvaluation::IO::CompressedRecordStore** (p. 269), **BiometricEvaluation::IO::ArchiveRecordStore** (p. 231), **BiometricEvaluation::IO::ListRecordStore** (p. 416), and

BiometricEvaluation::IO::SQLiteRecordStore (p. 554).

G.118.4 Member Data Documentation

G.118.4.1 BE_RECSTORE_SEQ_NEXT

`const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_NEXT = 2 [static]`

Tell sequence to sequence from current position

G.118.4.2 BE_RECSTORE_SEQ_START

`const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_START = 1 [static]`

Tell **sequence()** (p. 511) to sequence from beginning

G.118.4.3 INVALIDKEYCHARS

`const std::string BiometricEvaluation::IO::RecordStore::INVALIDKEYCHARS [static]`

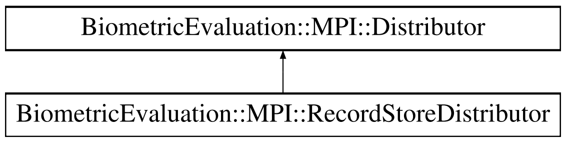
The set of prohibited characters in a key: '/', '\', '*', '&'

G.119 BiometricEvaluation::MPI::RecordStoreDistributor Class Reference

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

`#include <be_mpi_recordstoredistributor.h>`

Inheritance diagram for BiometricEvaluation::MPI::RecordStoreDistributor:


BiometricEvaluation::MPI::Distributor
BiometricEvaluation::MPI::RecordStoreDistributor

Public Member Functions

- **RecordStoreDistributor** (const std::string &propertiesFileName, const bool includeValues)

  Construct a distributor using the named properties.

Protected Member Functions

- void **createWorkPackage** (**MPI::WorkPackage** &workPackage)

  Create a work package for distribution.

G.119.1 Detailed Description

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

G.119.2 Constructor & Destructor Documentation

G.119.2.1 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. *307*)	An error occurred, typically due to missing or invalid properties.

See also

MPI::Distributor (p. 303)
MPI::RecordProcessor (p. 496)
MPI::RecordStoreResources (p. 519)

G.119.3 Member Function Documentation

G.119.3.1 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. 304).

G.120 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 postfix)
- **RecordStoreIterator operator+=** (**difference_type** rhs)

  Advance a variable number of arguments.
- **RecordStoreIterator operator+** (**difference_type** rhs)

  Advance a variable number of arguments.
- bool **operator==** (const **RecordStoreIterator** &rhs)

  Equivalence operator.
- bool **operator!=** (const **RecordStoreIterator** &rhs)

  Non-equivalence operator.
- **RecordStoreIterator** & **operator=** (**RecordStoreIterator** &rhs)=default
- **RecordStoreIterator** & **operator=** (**RecordStoreIterator** &&rhs)=default

G.120.1 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. 500).
This generic iterator provides no optimization over **RecordStore::sequence()** (p. 511).

G.120.2 Member Typedef Documentation

G.120.2.1 difference_type

```
using  BiometricEvaluation::IO::RecordStoreIterator::difference_type = std::ptrdiff_t
```

Type used to measure distance between iterators

G.120.2.2 iterator_category

```
using  BiometricEvaluation::IO::RecordStoreIterator::iterator_category = std::forward_iterator←↩
_tag
```

Type of iterator

G.120.2.3 pointer

```
using  BiometricEvaluation::IO::RecordStoreIterator::pointer =  value_type*
```

Pointer to the type iterated over

G.120.2.4 reference

```
using  BiometricEvaluation::IO::RecordStoreIterator::reference =  value_type&
```

Reference to the type iterated over

G.120.2.5 value_type

```
using  BiometricEvaluation::IO::RecordStoreIterator::value_type =  RecordStore::Record
```

Type when dereferencing iterators

G.120.3 Constructor & Destructor Documentation

G.120.3.1 RecordStoreIterator() [1/4]

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

Default constructor.

Creates ”end” iterator.

Note

Satisfies DefaultConstructible requirement.

G.120.3.2 RecordStoreIterator() [2/4]

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

Constructor.

Parameters

recordStore	Pointer to a **RecordStore** (p. 500) 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. 500).
RecordStoreIterator (p. 514) does not retain any ownership of recordStore.

G.120.3.3 RecordStoreIterator() [3/4]

```
BiometricEvaluation::IO::RecordStoreIterator::RecordStoreIterator (
             const  RecordStoreIterator & rhs )  [default]
```

Default copy constructor

G.120.3.4 RecordStoreIterator() [4/4]

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

Default move constructor

G.120.3.5 ∼RecordStoreIterator()

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

Default destructor

G.120.4 Member Function Documentation

G.120.4.1 operator"!=()

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

Non-equivalence operator.

Parameters

rhs	Reference to **RecordStoreIterator** (p. 514) being compared.

Returns

Whether or not this is not equivalent to rhs.

Note

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

G.120.4.2 operator∗()

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

Returns

Reference to a Record.

G.120.4.3 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.

G.120.4.4 operator++() [1/2]

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

Returns

Self after advancing.

G.120.4.5 operator++() [2/2]

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

Returns

Copy of self before advancing.

G.120.4.6 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.

G.120.4.7 operator->()

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

Returns

A dereferenced Record.

G.120.4.8 operator=()

```
RecordStoreIterator& BiometricEvaluation::IO::RecordStoreIterator::operator= (
            RecordStoreIterator && rhs )  [default]
```

Default move assignment operator

G.120.4.9 operator==()

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

Equivalence operator.

Parameters

rhs	Reference to **RecordStoreIterator** (p. 514) being compared.

Returns

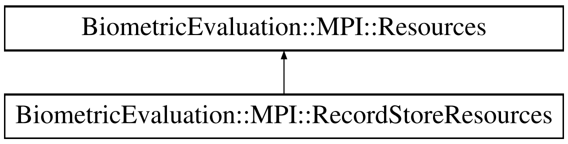
Whether or not this is equivalent to rhs.

G.121 BiometricEvaluation::MPI::RecordStoreResources Class Reference

A class to represent a set of resources needed by an **MPI** (p. 143) program using a RecordStore for input.

`#include <be_mpi_recordstoreresources.h>`

Inheritance diagram for BiometricEvaluation::MPI::RecordStoreResources:


BiometricEvaluation::MPI::Resources
BiometricEvaluation::MPI::RecordStoreResources

Public Member Functions

- **RecordStoreResources** (const std::string &propertiesFileName)

  Constructor taking the name of the properties file with the resource names.
- uint32_t **getChunkSize** () const
- bool **haveRecordStore** () const

  Indicator that a record store has been opened.
- std::shared_ptr< **IO::RecordStore** > **getRecordStore** () const

  Return the RecordStore named in the property set.

Static Public Member Functions

- static std::vector< std::string > **getRequiredProperties** ()

  Obtain the required properties as strings.
- static std::vector< std::string > **getOptionalProperties** ()

  Obtain the list of optional properties.

Static Public Attributes

- static const std::string **INPUTRSPROPERTY**

  The property string "Input Record Store"; required.
- static const std::string **CHUNKSIZEPROPERTY**

  The property string "Chunk Size"; required.

G.121.1 Detailed Description

A class to represent a set of resources needed by an **MPI** (p. 143) program using a RecordStore for input.

Resources (p. 528) are opened based on the property when appropriate. The input record store need not be accessible. Applications should call **haveRecordStore()** (p. 521) to check whether the record store has been opened.

G.121.2 Constructor & Destructor Documentation

G.121.2.1 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. *312*)	The resources file could not be read.
Error::ObjectDoesNotExist (p. *453*)	A required property does not exist.
Error::Exception (p. *307*)	Some other error occurred.

G.121.3 Member Function Documentation

G.121.3.1 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.

G.121.3.2 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.

G.121.3.3 getRequiredProperties()

```
static std::vector<std::string> BiometricEvaluation::MPI::RecordStoreResources::getRequired↩
Properties ( ) [static]
```

Obtain the required properties as strings.

Returns

The set of required properties.

G.121.3.4 haveRecordStore()

```
bool BiometricEvaluation::MPI::RecordStoreResources::haveRecordStore ( ) const
```

Indicator that a record store has been opened.

Returns

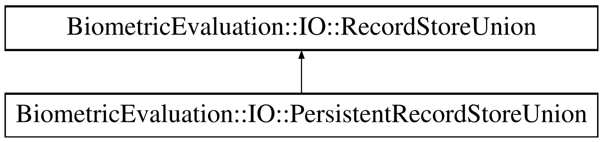
true if input record store is opened, false otherwise.

G.122 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:


BiometricEvaluation::IO::RecordStoreUnion
BiometricEvaluation::IO::PersistentRecordStoreUnion

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::I↩ O::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< **Biometric↩ Evaluation::IO::RecordStore** >>> recordStores)
- std::shared_ptr< **BiometricEvaluation::IO::RecordStore** > **getRecordStore** (const std::string &name) const

  Obtain a pointer to an open ***RecordStore*** *(p. 500).*
- 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.

G.122.1 Detailed Description

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

A **RecordStoreUnion** (p. 521) object is not copyable due to the fact that most **RecordStore** (p. 500) objects are not copyable.

G.122.2 Constructor & Destructor Documentation

G.122.2.1 RecordStoreUnion() [1/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion (
            const std::map< const std::string, const std::string > & recordStores )
```

RecordStoreUnion (p. 521) constructor.

Parameters

recordStores	Map of developer-provided names to paths to a **RecordStore** (p. 500).

G.122.2.2 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. 521) constructor.

Parameters

first	Iterator to the start of a map of developer-provided names to paths to a **RecordStore** (p. 500).
last	Iterator to the end of a map of developer-provided names to paths to a **RecordStore** (p. 500).

G.122.2.3 RecordStoreUnion() [3/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion (
             std::initializer_list< std::pair< const std::string, const std::string >> record↩
Stores )
```

RecordStoreUnion (p. 521) constructor.

Parameters

recordStores	List of pairs of developer-provided name and paths to a **RecordStore** (p. 500).

G.122.2.4 RecordStoreUnion() [4/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion (
             const std::map< const std::string, const std::shared_ptr< BiometricEvaluation↩
::IO::RecordStore >> & recordStores )
```

RecordStoreUnion (p. 521) constructor.

Parameters

recordStores	Map of developer-provided names and open **RecordStore** (p. 500) objects.

Note

Behavior when providing a **RecordStore** (p. 500) that has been opened read/write is undefined.

G.122.2.5 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. 521) constructor.

Parameters

first	Iterator to the start of a map of developer-provided names and open **RecordStore** (p. 500) objects.
last	Iterator to the end of a map of developer-provided names and open **RecordStore** (p. 500) objects.

Note

Behavior when providing a **RecordStore** (p. 500) that has been opened read/write is undefined.

G.122.2.6 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. 521) constructor.

Parameters

recordStores	List of pairs of developer-provided name and open **RecordStore** (p. 500) objects.

Note

Behavior when providing a **RecordStore** (p. 500) that has been opened read/write is undefined.

G.122.2.7 ∼RecordStoreUnion()

```
BiometricEvaluation::IO::RecordStoreUnion::∼RecordStoreUnion ( )
```

Destructor.

G.122.2.8 RecordStoreUnion() [7/7]

```
BiometricEvaluation::IO::RecordStoreUnion::RecordStoreUnion ( )  [protected]
```

Empty constructor for children.

Note

Implementation is not set. Callers must also call **setImpl()** (p. 526) to provide functionality.

setImpl

G.122.3 Member Function Documentation

G.122.3.1 getNames()

```
std::vector<std::string> BiometricEvaluation::IO::RecordStoreUnion::getNames ( ) const
```

Obtain the names of RecordStores set during construction.

Returns

Vector of names of RecordStores.

G.122.3.2 getRecordStore()

```
std::shared_ptr< BiometricEvaluation::IO::RecordStore> BiometricEvaluation::IO::RecordStore↩
Union::getRecordStore (
            const std::string & name ) const
```

Obtain a pointer to an open **RecordStore** (p. 500).

Parameters

name	Name provided to **RecordStore** (p. 500) during construction.

Exceptions

ObjectDoesNotExist	name is not recognized.

G.122.3.3 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. 500) name to data length read from said **RecordStore** (p. 500).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	key does not exist in any member RecordStores.
Error::StrategyError *(p. 560)*	Exceptions propagated from **RecordStore** (p. 500), with the exception of ObjectDoesNotExist.

Note

Exceptions are thrown after **length()** (p. 525) has been called on all member RecordStores.

G.122.3.4 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. 500) name to data read from said **RecordStore** (p. 500).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	key does not exist in any member RecordStores.
Error::StrategyError *(p. 560)*	Exceptions propagated from **RecordStore** (p. 500), with the exception of ObjectDoesNotExist.

Note

Exceptions are thrown after **read()** (p. 525) has been called on all member RecordStores.

G.122.3.5 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.

G.123 BiometricEvaluation::Image::Resolution Struct Reference

A structure to represent the resolution of an image.

`#include <be_image.h>`

Public Types

- enum **Units** { **Units::NA** = 0, **Units::PPI** = 1, **Units::PPMM** = 2, **Units::PPCM** = 3 }

  *Possible representations of the units in a **Resolution** (p. 526) struct.*

Public Member Functions

- **Resolution** (const double **xRes**=0.0, const double **yRes**=0.0, const **Units units**= **Units::PPI**)

  Create a ***Resolution*** *(p. 526) struct.*

- **Resolution toUnits** (const **Units** & **units**) const

  Obtain alternate representations of this resolution.

Public Attributes

- double **xRes**
- double **yRes**
- **Units units**

G.123.1 Detailed Description

A structure to represent the resolution of an image.

G.123.2 Member Enumeration Documentation

G.123.2.1 Units

```
enum BiometricEvaluation::Image::Resolution::Units [strong]
```

Possible representations of the units in a **Resolution** (p. 526) struct.

Enumerator

NA	Not-applicable: unknown, or otherwise
PPI	Pixels per inch
PPMM	Pixels per millimeter
PPCM	Pixels per centimeter

G.123.3 Constructor & Destructor Documentation

G.123.3.1 Resolution()

```
BiometricEvaluation::Image::Resolution::Resolution (
            const double xRes = 0.0,
            const double yRes = 0.0,
            const Units units = Units::PPI )
```

Create a **Resolution** (p. 526) struct.

Parameters

in	*xRes*	**Resolution** (p. 526) along the X-axis
in	*yRes*	**Resolution** (p. 526) along the Y-axis
in	*units*	Units in which xRes and yRes are represented

G.123.4 Member Function Documentation

G.123.4.1 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.

G.123.5 Member Data Documentation

G.123.5.1 units

```
Units BiometricEvaluation::Image::Resolution::units
```

Units in which xRes and yRes are represented

G.123.5.2 xRes

```
double BiometricEvaluation::Image::Resolution::xRes
```

Resolution (p. 526) along the X-axis

G.123.5.3 yRes

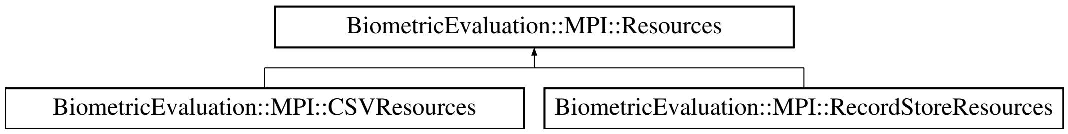
```
double BiometricEvaluation::Image::Resolution::yRes
```

Resolution (p. 526) along the Y-axis

G.124 BiometricEvaluation::MPI::Resources Class Reference

```
#include <be_mpi_resources.h>
```

Inheritance diagram for BiometricEvaluation::MPI::Resources:


BiometricEvaluation::MPI::Resources
BiometricEvaluation::MPI::CSVResources
BiometricEvaluation::MPI::RecordStoreResources

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. 124):Logsheet object.*
- int **getRank** () const
- int **getNumTasks** () const
- int **getWorkersPerNode** () const

Static Public Member Functions

- static std::vector< std::string > **getRequiredProperties** ()

  Obtain the list of required properties.
- static std::vector< std::string > **getOptionalProperties** ()

  Obtain the list of optional properties.

Static Public Attributes

- static const std::string **WORKERSPERNODEPROPERTY**

  The property string “Workers Per Node"; required.
- static const std::string **LOGSHEETURLPROPERTY**

  The property string “Logsheet URL"; optional.

G.124.1 Detailed Description

A class to represent a set of resources needed by an **MPI** (p. 143) program. The resources are based on a properties file as well as some dynamic information, such as **MPI** (p. 143) rank and process ID.

G.124.2 Constructor & Destructor Documentation

G.124.2.1 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. 312)*	The resources file could not be read.
Error::ObjectDoesNotExist *(p. 453)*	A required property does not exist.
Error::Exception *(p. 307)*	Some other error occurred.

G.124.3 Member Function Documentation

G.124.3.1 getLogsheetURL()

```
std::string BiometricEvaluation::MPI::Resources::getLogsheetURL ( ) const
```

Obtain the Uniform Resource Locator for the **IO** (p. 124):Logsheet object.

This string my be empty, indicating that there is no Logsheet URL in the Properties file.

Returns

The Logsheet URL.

G.124.3.2 getOptionalProperties()

```
static std::vector<std::string> BiometricEvaluation::MPI::Resources::getOptionalProperties (
) [static]
```

Obtain the list of optional properties.

Returns

A set of optional property strings.

G.124.3.3 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.

G.124.3.4 getRequiredProperties()

```
static std::vector<std::string> BiometricEvaluation::MPI::Resources::getRequiredProperties (
) [static]
```

Obtain the list of required properties.

Returns

A set of required property strings.

G.125 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.

Public Attributes

- uint64_t **elapsed**
- T **status**

  Value returned from operation.
- **APICurrentState currentState**

  Current state of operation.

G.125.1 Detailed Description

template<typename T>
class BiometricEvaluation::Framework::API< T >::Result

The result of an operation.

G.125.2 Constructor & Destructor Documentation

G.125.2.1 Result()

```
template<typename T >
BiometricEvaluation::Framework::API< T >::Result::Result ( )
```

Constructor

G.125.3 Member Function Documentation

G.125.3.1 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.

G.125.3.2 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.

G.125.4 Member Data Documentation

G.125.4.1 elapsed

```
template<typename T >
uint64_t  BiometricEvaluation::Framework::API< T >::Result::elapsed
```

Time (p. 159) elapsed while calling operation.

G.125.4.2 status

```
template<typename T >
T  BiometricEvaluation::Framework::API< T >::Result::status
```

Value returned from operation.

Note

Only populated when currentState == **APICurrentState::Completed** (p. 116).

G.126 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. 532) struct.*

Public Attributes

- **RidgeCountExtractionMethod extraction_method**
- int **index_one**
- int **index_two**
- int **count**

G.126.1 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.

G.127 BiometricEvaluation::Image::ROI Struct Reference

A structure to represent a region of interest (**ROI** (p. 533)), 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**

G.127.1 Detailed Description

A structure to represent a region of interest (**ROI** (p. 533)), which is a bounding box and a set of coordinates.

G.127.2 Constructor & Destructor Documentation

G.127.2.1 ROI() [1/2]

```
BiometricEvaluation::Image::ROI::ROI ( )
```

Create an empty **ROI** (p. 533) object.

G.127.2.2 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. 533) 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.
in	*vertOffset*	The vertical offset of the region of interest.
in	*path*	The path offset of the region of interest.

G.128 BiometricEvaluation::MPI::Runtime Class Reference

Runtime (p. 533) support for the startup/shutdown of **MPI** (p. 143) jobs.

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

Public Member Functions

- **Runtime** (int &argc, char ∗∗&argv)

  *Construct the runtime environment for the processes making up the **MPI** (p. 143) job.*
- void **start** (**BiometricEvaluation::MPI::Distributor** &distributor, **BiometricEvaluation::MPI::**↩ **Receiver** &receiver)

  *Startup the runtime environment for the **MPI** (p. 143) job.*
- void **shutdown** ()

  *Shutdown the runtime environment for the **MPI** (p. 143) job.*
- void **abort** (int errcode)

  *Abort the runtime the **MPI** (p. 143) job.*

G.128.1 Detailed Description

Runtime (p. 533) support for the startup/shutdown of **MPI** (p. 143) jobs.

This class provides methods that are used by applications to start and shutdown the **MPI** (p. 143) 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.

G.128.2 Constructor & Destructor Documentation

G.128.2.1 Runtime()

```
BiometricEvaluation::MPI::Runtime::Runtime (
             int & argc,
             char **& argv )
```

Construct the runtime environment for the processes making up the **MPI** (p. 143) job.

Parameters

in	*argc*	The argument count, taken from the command line passed to main().
in	*argv*	The argument vector, taken from the command line passed to main().

G.128.3 Member Function Documentation

G.128.3.1 abort()

```
void BiometricEvaluation::MPI::Runtime::abort (
             int errcode )
```

Abort the runtime the **MPI** (p. 143) job.

This method will cause the **MPI** (p. 143) 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. 143) runtime.

G.128.3.2 shutdown()

```
void BiometricEvaluation::MPI::Runtime::shutdown ( )
```

Shutdown the runtime environment for the **MPI** (p. 143) job.
This method must be called in order for the **MPI** (p. 143) runtime to cleanly exit.

G.128.3.3 start()

```
void BiometricEvaluation::MPI::Runtime::start (
              BiometricEvaluation::MPI::Distributor & distributor,
              BiometricEvaluation::MPI::Receiver & receiver )
```

Startup the runtime environment for the **MPI** (p. 143) job.

Parameters

in	*distributor*	The **Distributor** (p. 303) object that will form the basis of the first **MPI** (p. 143) task.
in	*receiver*	The **Receiver** (p. 494) object which will form the basis of **MPI** (p. 143) tasks 1..n.

G.129 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. 535).*

G.129.1 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. 535) object wraps a system resource, the **Semaphore** (p. 535) can be passed to other functions, or inherited across a fork boundary.

G.129.2 Constructor & Destructor Documentation

G.129.2.1 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.
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. 454)*	The semaphore already exists with the given name.
Error::StrategyError *(p. 560)*	An error occurred when creating the semaphore.

G.129.2.2 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. 453)*	A semaphore does not exist with the given name.
Error::StrategyError *(p. 560)*	An error occurred when creating the semaphore.

G.129.3 Member Function Documentation

G.129.3.1 getName()

```
std::string BiometricEvaluation::Process::Semaphore::getName ( )
```

Obtain the name of the **Semaphore** (p. 535).

Returns

The name of the Sempahore.

G.129.3.2 post()

```
void BiometricEvaluation::Process::Semaphore::post ( )
```

Post (increment) to the semaphore.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	The semaphore is no longer valid.
Error::StrategyError *(p. 560)*	**System** (p. 150) error obtaining the semaphore.

G.129.3.3 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.
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. 453)*	The semaphore is no longer valid.
Error::NotImplemented *(p. 452)*	Function is not implemented on the system. Applications should then call **wait()** (p. 538) or **trywait()** (p. 538).
Error::StrategyError *(p. 560)*	**System** (p. 150) error obtaining the semaphore.

G.129.3.4 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. 453)*	The semaphore is no longer valid.
Error::StrategyError *(p. 560)*	**System** (p. 150) error obtaining the semaphore.

G.129.3.5 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. 453)*	The semaphore is no longer valid.
Error::StrategyError *(p. 560)*	**System** (p. 150) error obtaining the semaphore.

G.130 BiometricEvaluation::Error::SignalManager Class Reference

A **SignalManager** (p. 539) object is used to handle signals that come from the operating system.

```
#include <be_error_signal_manager.h>
```

Public Member Functions

- **SignalManager** ()
- **SignalManager** (const sigset_t signalSet)
- void **setSignalSet** (const sigset_t signalSet)
- void **clearSignalSet** ()
- void **setDefaultSignalSet** ()
- bool **sigHandled** ()
- void **start** ()
- void **stop** ()
- void **setSigHandled** ()
- void **clearSigHandled** ()

Static Public Attributes

- static bool **_canSigJump**
- static sigjmp_buf **_sigJumpBuf**

G.130.1 Detailed Description

A **SignalManager** (p. 539) object is used to handle signals that come from the operating system.

Applications typically do not invoke most methods of a **SignalManager** (p. 539), except the **setSignal↩ Set()** (p. 541), **setDefaultSignalSet()** (p. 541), and **sigHandled()** (p. 541). An application wishing to just catch memory errors can simply construct a **SignalManager** (p. 539) object, and invoke **sigHandled()** (p. 541) 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. 539) object to start handling signals. Applications can call either **setSignalSet()** (p. 541) or **setDefaultSignalSet()** (p. 541) 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. 539) 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. 539) may be invoked, forcing a jump into an incompletely initialized function.

A **SignalManager** (p. 539) is passive (i.e. no signal handlers are installed) until that **start()** (p. 541) method is called, and becomes passive when **stop()** (p. 542) 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. 541) is called.

Attention

> The **start()** (p. 541), **stop()** (p. 542), **setSigHandled()** (p. 541) and **clearSigHandled()** (p. 540) methods are not meant to be used directly by applications, which should use the BEGIN_SIGNAL_BLOCK()/E↩ND_SIGNAL_BLOCK() macro pair.

G.130.2 Constructor & Destructor Documentation

G.130.2.1 SignalManager() `[1/2]`

```
BiometricEvaluation::Error::SignalManager::SignalManager ( )
```

Construct a new **SignalManager** (p. 539) object with the default signal handling: SIGSEGV and SIGBUS.

Exceptions

Error::StrategyError *(p. 560)*	Could not register the signal handler.

G.130.2.2 SignalManager() `[2/2]`

```
BiometricEvaluation::Error::SignalManager::SignalManager (
            const sigset_t signalSet )
```

Construct a new **SignalManager** (p. 539) 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. 470)*	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

G.130.3 Member Function Documentation

G.130.3.1 clearSigHandled()

```
void BiometricEvaluation::Error::SignalManager::clearSigHandled ( )
```

Clear the indication that a signal was handled.

G.130.3.2 clearSignalSet()

```
void BiometricEvaluation::Error::SignalManager::clearSignalSet ( )
```

Clear all signal handling.

G.130.3.3 setDefaultSignalSet()

```
void BiometricEvaluation::Error::SignalManager::setDefaultSignalSet ( )
```

Set the default signals this object will manage: SIGSEGV and SIGBUS.

G.130.3.4 setSigHandled()

```
void BiometricEvaluation::Error::SignalManager::setSigHandled ( )
```

Set a flag to indicate a signal was handled.

G.130.3.5 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. 470)*	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

G.130.3.6 sigHandled()

```
bool BiometricEvaluation::Error::SignalManager::sigHandled ( )
```

Indicate whether a signal was handled.

Returns

true if a signal was handled, false otherwise.

G.130.3.7 start()

```
void BiometricEvaluation::Error::SignalManager::start ( )
```

Start handling signals of the current signal set.

Exceptions

Error::StrategyError *(p. 560)*	Could not register the signal handler.

Note

If an application invokes **start()** (p. 541) 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.

G.130.3.8 stop()

```
void BiometricEvaluation::Error::SignalManager::stop ( )
```

Stop handling signals of the current signal set.

Exceptions

Error::StrategyError *(p. 560)*	Could not register the signal handler.

G.130.4 Member Data Documentation

G.130.4.1 _canSigJump

```
bool BiometricEvaluation::Error::SignalManager::_canSigJump [static]
```

Flag indicating can jump after handling a signal.

Note

Should not be directly used by applications.

G.130.4.2 _sigJumpBuf

```
sigjmp_buf BiometricEvaluation::Error::SignalManager::_sigJumpBuf [static]
```

The jump buffer used by the signal handler.

Note

Should not be directly used by applications.

G.131 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. 542) struct.*

Public Attributes

- uint32_t **xSize**
- uint32_t **ySize**

G.131.1 Detailed Description

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

G.131.2 Constructor & Destructor Documentation

G.131.2.1 Size()

```
BiometricEvaluation::Image::Size::Size (
            const uint32_t xSize = 0,
            const uint32_t ySize = 0 )
```

Create a **Size** (p. 542) struct.

Parameters

in	*xSize*	Number of pixels on the X-axis
in	*ySize*	Number of pixels on the Y-axis

G.131.3 Member Data Documentation

G.131.3.1 xSize

```
uint32_t BiometricEvaluation::Image::Size::xSize
```

Number of pixels on the X-axis

G.131.3.2 ySize

```
uint32_t BiometricEvaluation::Image::Size::ySize
```

Number of pixels on the Y-axis

G.132 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. 215) 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.

G.132.1 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.

G.132.2 Constructor & Destructor Documentation

G.132.2.1 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. 470)	No card exists for the given card number.
Error::StrategyError (p. 560)	Failed to access at least one of the readers.

G.132.2.2 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. 546) 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. 217)*	An error occurred activating the application. The status word fields on the exception's response object should be read to determine the error.
Error::ParameterError *(p. 470)*	No card exists for the given card number with the given application ID.
Error::StrategyError *(p. 560)*	Failed to access at least one of the readers.

G.132.2.3 ∼Smartcard()

```
BiometricEvaluation::Device::Smartcard::∼Smartcard ( )
```

Destructor.

G.132.2.4 Smartcard() [3/3]

```
BiometricEvaluation::Device::Smartcard::Smartcard (
               Smartcard && other )  [noexcept]
```

Move constructor.

Smartcard (p. 543) 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.

G.132.3 Member Function Documentation

G.132.3.1 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. 575) 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. 217)*	An error occurred activating the application. The status word fields on the exception's response object should be read to determine the error. The data field of the response may contain partial data from the card.
Error::StrategyError *(p. 560)*	An error occurred when communicating with the card.
Error::ParameterError *(p. 470)*	The object ID is too large.

G.132.3.2 getLastAPDU()

`Memory::uint8Array BiometricEvaluation::Device::Smartcard::getLastAPDU ( ) const`

Obtain a copy of the last **APDU** (p. 215) sent to the card.

Returns

The last sent **APDU** (p. 215) as an array of octets.

G.132.3.3 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.

G.132.3.4 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.

G.132.3.5 operator=()

```
Smartcard& BiometricEvaluation::Device::Smartcard::operator= (
             Smartcard && other ) [noexcept]
```

Move assignment.

Smartcard (p. 543) 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.

G.132.3.6 sendAPDU()

```
APDUResponse BiometricEvaluation::Device::Smartcard::sendAPDU (
             Device::Smartcard::APDU & apdu )
```

Send an **APDU** (p. 215) to a card using the best transmission method available for the card.

Parameters

in,out	*apdu*	The **APDU** (p. 215) to be sent. Fields may be modified by the function, specifically the length field(s).

Exceptions

APDUException *(p. 217)*	The status words from the command response are something other than 0x9000. The status word fields on the exception's response object should read to determine the result of the command. The data field of the response may contain partial data from the card.
Error::StrategyError *(p. 560)*	An error occurred when communicating with the card.

G.132.3.7 setDryrun()

```
void BiometricEvaluation::Device::Smartcard::setDryrun (
             bool state )
```

Set the 'dryrun' state.

Parameters

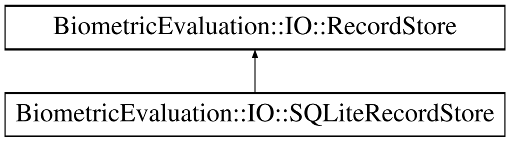
in	*state*	True when the **APDU** (p. 215) should be created, but not sent to the card. **getLastAPDU()** (p. 546)

G.133 BiometricEvaluation::IO::SQLiteRecordStore Class Reference

A **RecordStore** (p. 500) implementation using a SQLite database as the underlying record storage system.

`#include <be_io_sqliterecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::SQLiteRecordStore:


BiometricEvaluation::IO::RecordStore
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. 500).*
- 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. 500), returning the key/data pairs.*
- std::string **sequenceKey** (int cursor= **BE_RECSTORE_SEQ_NEXT**) override

  *Sequence through a **RecordStore** (p. 500), returning the key.*
- void **setCursorAtKey** (const std::string &key) override
- **SQLiteRecordStore** (const **SQLiteRecordStore** &)=delete
- **SQLiteRecordStore** & **operator=** (const **SQLiteRecordStore** &)=delete

Additional Inherited Members

G.133.1 Detailed Description

A **RecordStore** (p. 500) implementation using a SQLite database as the underlying record storage system.

G.133.2 Member Function Documentation

G.133.2.1 changeDescription()

```
void BiometricEvaluation::IO::SQLiteRecordStore::changeDescription (
            const std::string & description )  [override], [virtual]
```

Change the description of the **RecordStore** (p. 500).

Parameters

in	*description*	The new description.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 502).

G.133.2.2 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.133.2.3 getCount()

```
unsigned int BiometricEvaluation::IO::SQLiteRecordStore::getCount ( ) const  [override], [virtual]
```

Obtain the number of items in the **RecordStore** (p. 500).

Returns

The number of items in the **RecordStore** (p. 500).

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.133.2.4 getDescription()

```
std::string BiometricEvaluation::IO::SQLiteRecordStore::getDescription ( ) const  [override],
[virtual]
```

Obtain a textual description of the **RecordStore** (p. 500).

Returns

The **RecordStore** (p. 500)'s description.

Implements **BiometricEvaluation::IO::RecordStore** (p. 504).

G.133.2.5 getPathname()

```
std::string BiometricEvaluation::IO::SQLiteRecordStore::getPathname ( ) const  [override], [virtual]
```

Return the path name of the **RecordStore** (p. 500).

Returns

Where in the file system the **RecordStore** (p. 500) is located.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.133.2.6 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. 500).

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 505).

G.133.2.7 insert()

```
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.
in	*size*	The size of the record, in bytes.

Exceptions

Error::ObjectExists (p. *454*)	A record with the given key is already present.
Error::StrategyError (p. *560*)	The **RecordStore** (p. 500) is opened read-only, or an error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.133.2.8 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. *453*)	A record for the key does not exist.
Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 506).

G.133.2.9 move()

```
void BiometricEvaluation::IO::SQLiteRecordStore::move (
            const std::string & pathname )  [override], [virtual]
```

Move the **RecordStore** (p. 500).

The **RecordStore** (p. 500) can be moved to a new path in the file system.

Parameters

in	*pathname*	The new path of the **RecordStore** (p. 500).

Exceptions

Error::StrategyError (p. *560*)	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 507).

G.133.2.10 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 508).

G.133.2.11 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. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 509).

G.133.2.12 sequence()

```
RecordStore::Record BiometricEvaluation::IO::SQLiteRecordStore::sequence (
            int cursor = BE_RECSTORE_SEQ_NEXT )  [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.133.2.13 sequenceKey()

```
std::string BiometricEvaluation::IO::SQLiteRecordStore::sequenceKey (
            int cursor = BE_RECSTORE_SEQ_NEXT ) [override], [virtual]
```

Sequence through a **RecordStore** (p. 500), 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. 500) 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. 453)*	End of sequencing.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 511).

G.133.2.14 setCursorAtKey()

```
void BiometricEvaluation::IO::SQLiteRecordStore::setCursorAtKey (
            const std::string & key )  [override], [virtual]
```

Set the sequence cursor to an arbitrary position within the **RecordStore** (p. 500), starting at key. Key will be the first record returned from the next call to **sequence()** (p. 552).

Parameters

in	*key*	The key of the record which will be returned by the first subsequent call to **sequence()** (p. 552).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	A record for the key does not exist.
Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.133.2.15 sync()

```
void BiometricEvaluation::IO::SQLiteRecordStore::sync ( ) const  [override], [virtual]
```

Synchronize the entire record store to persistent storage.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when using the underlying storage system.

Implements **BiometricEvaluation::IO::RecordStore** (p. 512).

G.134 BiometricEvaluation::Process::Statistics Class Reference

The **Statistics** (p. 554) class provides an interface for gathering process statistics, such as memory usage, system time, etc.

`#include <be_process_statistics.h>`

Public Member Functions

- **Statistics** ()
- **Statistics** (**IO::FileLogCabinet** ∗const logCabinet)
- **Statistics** (const std::shared_ptr< **IO::Logsheet** > &logSheet)

  Construct a Statistic object that logs to an existing Logsheet.
- void **getCPUTimes** (uint64_t ∗usertime, uint64_t ∗systemtime)
- void **getMemorySizes** (uint64_t ∗vmrss, uint64_t ∗vmsize, uint64_t ∗vmpeak, uint64_t ∗vmdata, uint64←_t ∗vmstack)
- uint32_t **getNumThreads** ()
- void **logStats** ()

  Create a snapshot of the current process statistics in the FileLogsheet created in the FileLogCabinet.

- void **startAutoLogging** (uint64_t interval)

  Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.
- void **stopAutoLogging** ()

  Stop the automatic logging of process statistics.
- void **callStatistics_logStats** ()

G.134.1 Detailed Description

The **Statistics** (p. 554) class provides an interface for gathering process statistics, such as memory usage, system time, etc.

The information gathered by objects of this class are for the current process, and can optionally be logged to a FileLogsheet object contained within the provided FileLogCabinet.

Note

> The resolution of a returned value for many methods may not match the resolution allowed by the interface. For example, the operating system my allow for second resolution whereas the interface allows microsecond resolution.

G.134.2 Constructor & Destructor Documentation

G.134.2.1 Statistics() [1/3]

```
BiometricEvaluation::Process::Statistics::Statistics ( )
```

Constructor with no parameters.

G.134.2.2 Statistics() [2/3]

```
BiometricEvaluation::Process::Statistics::Statistics (
                IO::FileLogCabinet *const logCabinet )
```

Construct a **Statistics** (p. 554) object with the associated FileLogCabinet.

Parameters

in	*logCabinet*	The FileLogCabinet obejct where this object will create a FileLogsheet to contain the statistic information for the process.

Exceptions

Error::NotImplemented *(p. 452)*	Logging is not supported on this OS. This exception can be thrown when any portion of the statistics gathering cannot be completed.
Error::ObjectExists *(p. 454)*	The FileLogsheet already exists. This exception should rarely, if ever, occur.
Error::StrategyError *(p. 560)*	Failure to create the FileLogsheet in the cabinet.

G.134.2.3 Statistics() [3/3]

```
BiometricEvaluation::Process::Statistics::Statistics (
             const std::shared_ptr< IO::Logsheet > & logSheet )
```

Construct a Statistic object that logs to an existing Logsheet.

Parameters

in	*logSheet*	Existing Logsheet that will be appended.

Exceptions

Error::NotImplemented (p. *452*)	Logging is not supported on this OS. This exception can be thrown when any portion of the statistics gathering cannot be completed.

G.134.3 Member Function Documentation

G.134.3.1 callStatistics_logStats()

```
void BiometricEvaluation::Process::Statistics::callStatistics_logStats ( )
```

Helper function in C++ space that has access to this object, and is called from C space by the logging thread. Applications should not call this function.

G.134.3.2 getCPUTimes()

```
void BiometricEvaluation::Process::Statistics::getCPUTimes (
             uint64_t * usertime,
             uint64_t * systemtime )
```

Obtain the total user and system times for the process, in microseconds. Any of the out parameters can be nullptr, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

out	*usertime*	Pointer where to store the total user time.
out	*systemtime*	Pointer where to store the total system time.

Exceptions

Error::StrategyError (p. *560*)	An error occurred when obtaining the process statistics from the operating system. The exception information string contains the error reason.
Error::NotImplemented (p. *452*)	This method is not implemented on this OS.

G.134.3.3 getMemorySizes()

```
void BiometricEvaluation::Process::Statistics::getMemorySizes (
             uint64_t * vmrss,
             uint64_t * vmsize,
             uint64_t * vmpeak,
             uint64_t * vmdata,
             uint64_t * vmstack )
```

Obtain the current memory set sizes for the process, in kilobytes. Any of the out parameters can be nullptr, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

out	*vmrss*	Pointer where to store the current resident set size.
out	*vmsize*	Pointer where to store the current total virtual memory size.
out	*vmpeak*	Pointer where to store the peak total virtual memory size.
out	*vmdata*	Pointer where to store the current virtual memory data segment size.
out	*vmstack*	Pointer where to store the current virtual memory stack segment size.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when obtaining the process statistics from the operating system. The exception information string contains the error reason.
Error::NotImplemented *(p. 452)*	This method is not implemented on this OS.

G.134.3.4 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. 560)*	An error occurred when obtaining the process info from the operating system. The exception information string contains the error reason.
Error::NotImplemented *(p. 452)*	This method is not implemented on this OS.

G.134.3.5 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. 453)*	The FileLogsheet does not exist; this object was not created with FileLogCabinet object.
Error::StrategyError *(p. 560)*	An error occurred when writing to the FileLogsheet.
Error::NotImplemented *(p. 452)*	The statistics gathering is not implemented for this operating system.

G.134.3.6 startAutoLogging()

```
void BiometricEvaluation::Process::Statistics::startAutoLogging (
            uint64_t interval )
```

Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.

Note

It is unrealistic to expect that log entries can be made at a rate of one per microsecond.
If **stopAutoLogging()** (p. 558) is called very soon after the start, a log entry may not be made.

Parameters

`in`	*interval*	The gap between logging snapshots, in microseconds.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	The FileLogsheet does not exist; this object was not created with FileLogCabinet object.
Error::ObjectExists *(p. 454)*	Autologging is currently invoked.
Error::StrategyError *(p. 560)*	An error occurred when writing to the FileLogsheet.
Error::NotImplemented *(p. 452)*	The statistics gathering is not implemented for this operating system.

G.134.3.7 stopAutoLogging()

```
void BiometricEvaluation::Process::Statistics::stopAutoLogging ( )
```

Stop the automatic logging of process statistics.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	Not currently autologging.
Error::StrategyError *(p. 560)*	An error occurred when stopping, most likely because the logging thread died.

G.135 BiometricEvaluation::Framework::Status Class Reference

```
#include <be_framework_status.h>
```

Public Member Functions

- **Status** (int32_t code= **OK**, const std::string &message="") noexcept

  Status *(p. 559) constructor.*
- int32_t **getCode** () const noexcept

  Obtain the return code from this ***Status*** *(p. 559).*
- std::string **getMessage** () const noexcept

  Obtain the explanatory message from this ***Status*** *(p. 559).*

Static Public Attributes

- static const int32_t **OK** = 0

G.135.1 Detailed Description

Type to be returned from **API** (p. 220) methods

G.135.2 Constructor & Destructor Documentation

G.135.2.1 Status()

```
BiometricEvaluation::Framework::Status::Status (
            int32_t code = OK,
            const std::string & message = "" ) [noexcept]
```

Status (p. 559) constructor.

Parameters

code	Return code from a function or method.
message	Message providing insight into code's value.

G.135.3 Member Function Documentation

G.135.3.1 getCode()

`int32_t BiometricEvaluation::Framework::Status::getCode ( ) const [inline], [noexcept]`

Obtain the return code from this **Status** (p. 559).

Returns

Return code

G.135.3.2 getMessage()

`std::string BiometricEvaluation::Framework::Status::getMessage ( ) const [inline], [noexcept]`

Obtain the explanatory message from this **Status** (p. 559).

Returns

Explanator message.

Note

May be empty.

G.135.4 Member Data Documentation

G.135.4.1 OK

`const int32_t BiometricEvaluation::Framework::Status::OK = 0 [static]`

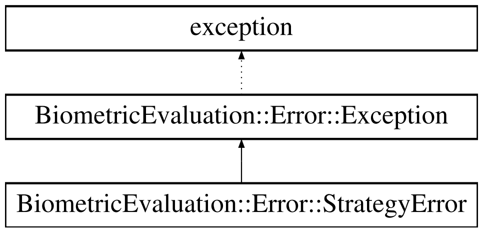
Successful return. Nothing to report.

G.136 BiometricEvaluation::Error::StrategyError Class Reference

A **StrategyError** (p. 560) object is thrown when the underlying implementation of this interface encounters an error.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::StrategyError:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::StrategyError

Public Member Functions

- **StrategyError** ()
- **StrategyError** (const std::string &info)

G.136.1 Detailed Description

A **StrategyError** (p. 560) object is thrown when the underlying implementation of this interface encounters an error.

G.136.2 Constructor & Destructor Documentation

G.136.2.1 StrategyError() [1/2]

```
BiometricEvaluation::Error::StrategyError::StrategyError ( )
```

Construct a **StrategyError** (p. 560) object with the default information string.

G.136.2.2 StrategyError() [2/2]

```
BiometricEvaluation::Error::StrategyError::StrategyError (
            const std::string & info )
```

Construct a **StrategyError** (p. 560) object with an information string appended to the default information string.

G.137 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.

G.137.1 Member Function Documentation

G.137.1.1 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.

G.137.1.2 getFrame()

```
virtual  Video::Frame BiometricEvaluation::Video::Stream::getFrame (
             uint32_t frameNum )  [pure virtual]
```

Obtain a frame from the video stream.

Parameters

frameNum	**Frame** (p. 342) number, >= 1

Exceptions

Error::ParameterError *(p. 470)*	frameNum is too large.
Error::StrategyError *(p. 560)*	No codec available for the video stream or other failure to read the stream.

G.137.1.3 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.

G.137.1.4 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, milliseconds.
endTime	Approximate time of the ending frame, milliseconds

Exceptions

Error::StrategyError *(p. 560)*	No codec available for the video stream or other failure to read the stream.

G.137.1.5 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.

G.137.1.6 setFrameScale()

```
virtual void BiometricEvaluation::Video::Stream::setFrameScale (
            float xScale,
            float yScale )  [pure virtual]
```

Set the scaling factors for returned video frames.

Parameters

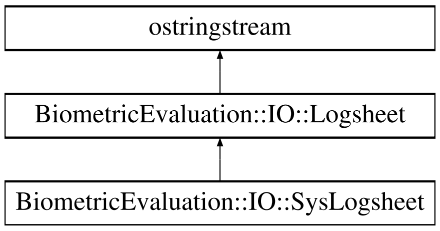
xScale	The scaling factor for frame width.
yScale	The scaling factor for frame height.

G.138 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:


ostringstream
BiometricEvaluation::IO::Logsheet
BiometricEvaluation::IO::SysLogsheet

Public Member Functions

- **SysLogsheet** (const std::string &url, const std::string &description, const std::string &appname, bool sequenced, bool utc)

  Create a new log sheet.
- **SysLogsheet** (const std::string &url, const std::string &description, const std::string &appname, const std::string &hostname, bool sequenced, bool utc)

  Create a new log sheet.
- ∼**SysLogsheet** ()
- void **write** (const std::string &entry)

Write a string as an entry to the backing store.

- void **writeComment** (const std::string &entry)

  Write a string as a comment to the backing store.
- void **writeDebug** (const std::string &entry)

  Write a string as a debug entry to the backing store.
- void **sync** ()

  Synchronize any buffered data to the underlying backing store.

Protected Member Functions

- **SysLogsheet** (const **SysLogsheet** &)
- **SysLogsheet** & **operator=** (const **SysLogsheet** &)
- void **setup** (const std::string &url, const std::string &description)
- void **writeToLogger** (const std::string &priority, const char delimiter, const std::string &prefix, const std::string &message)

Protected Attributes

- std::string **_hostname**
- std::string **_appname**
- std::string **_procid**
- int **_sockFD**
- bool **_sequenced**
- bool **_operational**
- bool **_utc**

Additional Inherited Members

G.138.1 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.

G.138.2 Constructor & Destructor Documentation

G.138.2.1 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.
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. 159) (UTC), false for local time.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when connecting to the logging system, or URL is malformed.

G.138.2.2 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.
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. 159) (UTC), false for local time.

Exceptions

Error::StrategyError *(p. 560)*	An error occurred when connecting to the logging system, or URL is malformed.

G.138.2.3 ∼SysLogsheet()

```
BiometricEvaluation::IO::SysLogsheet::∼SysLogsheet ( )
```

Destructor

G.138.2.4 SysLogsheet() [3/3]

```
BiometricEvaluation::IO::SysLogsheet::SysLogsheet (
             const  SysLogsheet &  )  [protected]
```

Prevent copying of **SysLogsheet** (p. 563) objects

G.138.3 Member Function Documentation

G.138.3.1 operator=()

```
SysLogsheet& BiometricEvaluation::IO::SysLogsheet::operator= (
             const  SysLogsheet &  )  [protected]
```

Prevent copying of **SysLogsheet** (p. 563) objects

G.138.3.2 setup()

```
void BiometricEvaluation::IO::SysLogsheet::setup (
             const std::string & url,
             const std::string & description )  [protected]
```

Helper function to build connections

G.138.3.3 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. 560)*	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 423).

G.138.3.4 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. 560)*	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 423).

G.138.3.5 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. 560)*	An error occurred when using the underlying backing store.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 424).

G.138.3.6 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. 560)*	An error occurred when logging.

Reimplemented from **BiometricEvaluation::IO::Logsheet** (p. 424).

G.138.3.7 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

G.138.4 Member Data Documentation

G.138.4.1 _operational

`bool BiometricEvaluation::IO::SysLogsheet::_operational [protected]`

Whether the sheet is operational

G.138.4.2 _sequenced

`bool BiometricEvaluation::IO::SysLogsheet::_sequenced [protected]`

Whether to include entry sequence numbers

G.138.4.3 _sockFD

`int BiometricEvaluation::IO::SysLogsheet::_sockFD [protected]`

Socket file descriptor for the logging system

G.138.4.4 _utc

`bool BiometricEvaluation::IO::SysLogsheet::_utc [protected]`

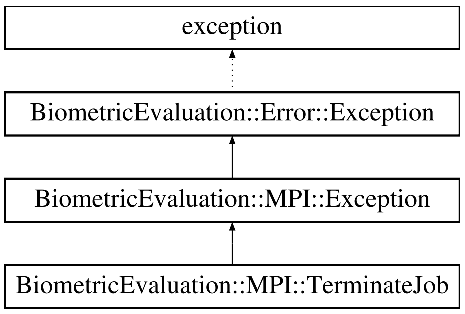
Whether time stamps are in UTC

G.139 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. 303).

`#include <be_mpi_exception.h>`

Inheritance diagram for BiometricEvaluation::MPI::TerminateJob:


exception
BiometricEvaluation::Error::Exception
BiometricEvaluation::MPI::Exception
BiometricEvaluation::MPI::TerminateJob

Public Member Functions

- **TerminateJob** ()
- **TerminateJob** (std::string info)

    Constructor.

G.139.1 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. 303).

G.139.2 Constructor & Destructor Documentation

G.139.2.1 TerminateJob() [1/2]

```
BiometricEvaluation::MPI::TerminateJob::TerminateJob ( )
```

Construct with default information string.

G.139.2.2 TerminateJob() [2/2]

```
BiometricEvaluation::MPI::TerminateJob::TerminateJob (
            std::string info )
```

Constructor.

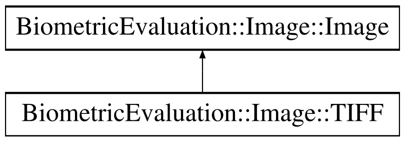
Parameters

info	Custom information string. Will be appended to the default information string.

G.140 BiometricEvaluation::Image::TIFF Class Reference

```
#include <be_image_tiff.h>
```

Inheritance diagram for BiometricEvaluation::Image::TIFF:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::TIFF

Public Member Functions

- **TIFF** (const uint8_t ∗data, const uint64_t size)
- **TIFF** (const **Memory::uint8Array** &data)
- **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.

Static Public Member Functions

- static bool **isTIFF** (const uint8_t *data, const uint64_t size)

  *Determine if image is encoded as **TIFF** (p. 569).*
- static bool **isTIFF** (const **Memory::uint8Array** &data)

  *Determine if image is encoded as **TIFF** (p. 569).*

Additional Inherited Members

G.140.1 Detailed Description

A TIFF-encoded image.

G.140.2 Member Function Documentation

G.140.2.1 getRawData()

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

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

Error::DataError (p. 293)	**Error** (p. 106) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 357).

G.140.2.2 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. 293)	**Error** (p. 106) decompressing image data.
Error::NotImplemented (p. 452)	Unsupported conversion based on source color depth.

Exceptions

Error::ParameterError *(p. 470)*	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. 358).

G.140.2.3 isTIFF() [1/2]

```
static bool BiometricEvaluation::Image::TIFF::isTIFF (
             const uint8_t * data,
             const uint64_t size )  [static]
```

Determine if image is encoded as **TIFF** (p. 569).

Parameters

in	*data*	**Image** (p. 351) data.
in	*size*	**Size** (p. 542) of `data`.

Returns

true if data appears to be encoded with **TIFF** (p. 569), false otherwise.

G.140.2.4 isTIFF() [2/2]

```
static bool BiometricEvaluation::Image::TIFF::isTIFF (
             const  Memory::uint8Array & data )  [static]
```

Determine if image is encoded as **TIFF** (p. 569).

Parameters

in	*data*	**Image** (p. 351) data.

Returns

true if data appears to be encoded with **TIFF** (p. 569), false otherwise.

G.141 BiometricEvaluation::Time::Timer Class Reference

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

```
#include <be_time_timer.h>
```

Public Types

- using **BE_CLOCK_TYPE** = std::chrono::steady_clock

Public Member Functions

- **Timer** ()
- **Timer** (const std::function< void()> &func)

    Construct a timer and time a function immediately.
- void **start** ()

    Start tracking time.
- void **stop** ()

    Stop tracking time.
- uint64_t **elapsed** (bool nano=false) const

    *Get the elapsed time in microseconds or nanoseconds between calls to this object's **start()** (p. 574) and **stop()** (p. 574) methods.*
- std::string **elapsedStr** (bool displayUnits=false, bool nano=false) const

    Convenience method for printing elapsed time as a string.
- **Timer** & **time** (const std::function< void()> &func)

    Record the runtime of a function.

G.141.1 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. 574) and **Timer::stop()** (p. 574) calls, then use **Timer::elapsed()** (p. 573) 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.

G.141.2 Member Typedef Documentation

G.141.2.1 BE_CLOCK_TYPE

```
using BiometricEvaluation::Time::Timer::BE_CLOCK_TYPE = std::chrono::steady_clock
```

Clock type to use, aliased for easy replacement.

G.141.3 Constructor & Destructor Documentation

G.141.3.1 Timer() [1/2]

```
BiometricEvaluation::Time::Timer::Timer ( )
```

Constructor for the **Timer** (p. 571) object.

G.141.3.2 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. 560)	Propagated from **time()** (p. 574).

G.141.4 Member Function Documentation

G.141.4.1 elapsed()

```
uint64_t BiometricEvaluation::Time::Timer::elapsed (
              bool nano = false ) const
```

Get the elapsed time in microseconds or nanoseconds between calls to this object's **start()** (p. 574) and **stop()** (p. 574) methods.

Parameters

nano	True if to return nanoseconds, false otherwise.

Returns

The number of microseconds or nanoseconds.

Exceptions

Error::StrategyError (p. 560)	This object is currently timing an operation or an error occurred when obtaining timing information.

G.141.4.2 elapsedStr()

```
std::string BiometricEvaluation::Time::Timer::elapsedStr (
              bool displayUnits = false,
              bool nano = false ) const
```

Convenience method for printing elapsed time as a string.

Parameters

displayUnits	Append the elapsed time units.
nano	True if to return nanoseconds, false otherwise.

Returns

String representing the elapsed time.

Exceptions

Error::StrategyError *(p. 560)*	Propagated from **elapsed()** (p. 573).

G.141.4.3 start()

```
void BiometricEvaluation::Time::Timer::start ( )
```

Start tracking time.

Exceptions

Error::StrategyError *(p. 560)*	This object is currently timing an operation or an error occurred when obtaining timing information.

G.141.4.4 stop()

```
void BiometricEvaluation::Time::Timer::stop ( )
```

Stop tracking time.

Exceptions

Error::StrategyError *(p. 560)*	This object is not currently timing an operation or an error occurred when obtaining timing information.

G.141.4.5 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. 560)*	Propagated from **start()** (p. 574) or **stop()** (p. 574), and/or func is nullptr.

G.142 BiometricEvaluation::Device::TLV Class Reference

A class to represent a Tag-Length-Value (**TLV** (p. 575)) 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. 575) 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. 575).*
- **Memory::uint8Array** **getPrimitive** () const

  *Obtain the primitive data associated with this **TLV** (p. 575).*
- void **addChild** (const **TLV** &tlv)
- std::vector< **TLV** > **getChildren** () const
- **Memory::uint8Array** **getRawTLV** () const

  *Obtain the **TLV** (p. 575) 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. 575) into a string object, in readable format.*

G.142.1 Detailed Description

A class to represent a Tag-Length-Value (**TLV** (p. 575)) data structure as described in the ISO 7816-4 integrated circuit card standard.

A **TLV** (p. 575) is composed of tag and length fields, then a value field that may be another **TLV** (p. 575) (a child), or data of another format, represented as the primitive object in this class.

G.142.2 Constructor & Destructor Documentation

G.142.2.1 TLV() [1/4]

```
BiometricEvaluation::Device::TLV::TLV ( )
```

Construct an empty Tag-Length-Value object that can be filled with setter methods.

Empty **TLV** (p. 575) objects are primitive.

G.142.2.2 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. 293)*	The data in the buffer is not conforming.

G.142.2.3 TLV() [3/4]

```
BiometricEvaluation::Device::TLV::TLV (
             Memory::IndexedBuffer & ibuf )
```

Construct a single **TLV** (p. 575) from the indexed buffer.

Exceptions

Error::DataError *(p. 293)*	**Error** (p. 106) parsing the data in the buffer.

G.142.2.4 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. 293)*	The data in the file is not conformance.

G.142.3 Member Function Documentation

G.142.3.1 addChild()

```
void BiometricEvaluation::Device::TLV::addChild (
             const  TLV & tlv )
```

Add a child **TLV** (p. 575).

Parameters

tlv	The **TLV** (p. 575) to be added as a child of this **TLV** (p. 575).

Exceptions

Error::DataError *(p. 293)*	The **TLV** (p. 575) is primitive.

G.142.3.2 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. 293)*	The **TLV** (p. 575) is primitive.

G.142.3.3 getPrimitive()

```
Memory::uint8Array BiometricEvaluation::Device::TLV::getPrimitive ( ) const
```

Obtain the primitive data associated with this **TLV** (p. 575).

Exceptions

Error::DataError *(p. 293)*	The **TLV** (p. 575) is of the constructed form.

See also

getChildren (p. 577).

G.142.3.4 getRawTLV()

`Memory::uint8Array BiometricEvaluation::Device::TLV::getRawTLV ( ) const`

Obtain the **TLV** (p. 575) 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. 575) as an array.

G.142.3.5 getTagClass()

`uint8_t BiometricEvaluation::Device::TLV::getTagClass ( ) const`

Get the decoded tag class.

Returns

The tag class.

G.142.3.6 getTagNum()

`uint32_t BiometricEvaluation::Device::TLV::getTagNum ( ) const`

Get the decoded tag number.

Returns

The tag number.

G.142.3.7 isPrimitive()

`bool BiometricEvaluation::Device::TLV::isPrimitive ( ) const`

Obtain the type of **TLV** (p. 575): primitive/constructed.

Returns

True if is a primitive **TLV** (p. 575), false otherwise.

G.142.3.8 setPrimitive()

```
void BiometricEvaluation::Device::TLV::setPrimitive (
              const Memory::uint8Array & value )
```

Set the primitive data associated with this **TLV** (p. 575).

The primitive data is added as the value data item.

Exceptions

Error::DataError *(p. 293)*	The **TLV** (p. 575) is already of the constructed form, meaning that there are **TLV** (p. 575) children set as the value data.

G.142.3.9 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. 293)*	The primitive indicator conflicts with the presence of children TLVs, or presence of primitive data.
Error::ParameterError *(p. 470)*	The length of the buffer is larger than the maximum tag length.

G.142.3.10 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. 575) into a string object, in readable format.

Parameters

tlv	The **TLV** (p. 575) to print.
tabCount	The number of tab characters to insert before each line of the output.

G.143 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 >

G.143.1 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.

G.143.2 Member Typedef Documentation

G.143.2.1 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.

G.144 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[ ]>

G.144.1 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.

G.144.2 Member Typedef Documentation

G.144.2.1 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.

G.145 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

G.145.1 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.

G.145.2 Member Typedef Documentation

G.145.2.1 unique_array_known_bound

```
template<class T , size_t S>
using  BiometricEvaluation::Memory::unique_if< T[S]>::  unique_array_known_bound = void
```

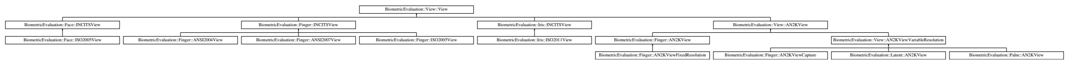
Type to use when T is known-bound array.

G.146 BiometricEvaluation::View::View Class Reference

A class to represent single biometric element view.

`#include <be_view_view.h>`

Inheritance diagram for BiometricEvaluation::View::View:


BiometricEvaluation::View::View
BiometricEvaluation::Face::INCITSView
BiometricEvaluation::Finger::INCITSView
BiometricEvaluation::Iris::INCITSView
BiometricEvaluation::View::AN2KView
BiometricEvaluation::Face::ISO2005View
BiometricEvaluation::Finger::ANSI2004View
BiometricEvaluation::Finger::ANSI2007View
BiometricEvaluation::Finger::ISO2005View
BiometricEvaluation::Iris::ISO2011View
BiometricEvaluation::Finger::AN2KView
BiometricEvaluation::View::AN2KViewVariableResolution
BiometricEvaluation::Finger::AN2KViewFixedResolution
BiometricEvaluation::Finger::AN2KViewCapture
BiometricEvaluation::Latent::AN2KView
BiometricEvaluation::Palm::AN2KView

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.

G.146.1 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.

G.146.2 Member Function Documentation

G.146.2.1 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.

G.146.2.2 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.

G.146.2.3 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.

G.146.2.4 getImageResolution()

`Image::Resolution BiometricEvaluation::View::View::getImageResolution ( ) const`

Obtain the image resolution.

Image (p. 118) 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. 527) field for value NA.

G.146.2.5 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.

G.146.2.6 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. 527) field for value NA.

G.146.2.7 setImageColorDepth()

```
void BiometricEvaluation::View::View::setImageColorDepth (
            uint32_t imageColorDepth )  [protected]
```

Mutator for the image color depth.

Parameters

in	*imageColorDepth*	The image color depth.

G.146.2.8 setImageData()

```
void BiometricEvaluation::View::View::setImageData (
            const  BiometricEvaluation::Memory::uint8Array & imageData )  [protected]
```

Mutator for the image data.

Parameters

in	*imageData*	The image data object.

G.146.2.9 setImageResolution()

```
void BiometricEvaluation::View::View::setImageResolution (
            const  BiometricEvaluation::Image::Resolution & imageResolution )  [protected]
```

Mutator for the image resolution.

Parameters

in	*imageResolution*	The image resolution object.

G.146.2.10 setImageSize()

```
void BiometricEvaluation::View::View::setImageSize (
            const  BiometricEvaluation::Image::Size & imageSize )  [protected]
```

Mutator for the image size.

Parameters

in	*imageSize*	The image size object.

G.146.2.11 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.

G.147 BiometricEvaluation::Time::Watchdog Class Reference

A **Watchdog** (p. 585) object can be used by applications to limit the amount of processing time taken by a block of code.

```
#include <be_time_watchdog.h>
```

Public Member Functions

- **Watchdog** (const uint8_t type)
- void **setInterval** (uint64_t interval)
- void **start** ()
- void **stop** ()
- bool **expired** ()
- void **setCanSigJump** ()
- void **clearCanSigJump** ()
- void **setExpired** ()
- void **clearExpired** ()

Static Public Attributes

- static const uint8_t **PROCESSTIME** = 0
- static const uint8_t **REALTIME** = 1
- static bool **_canSigJump**
- static sigjmp_buf **_sigJumpBuf**

G.147.1 Detailed Description

A **Watchdog** (p. 585) object can be used by applications to limit the amount of processing time taken by a block of code.

A **Watchdog** (p. 585) 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. 585) builds on the POSIX setitimer(2) call. **Timer** (p. 571) 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 BEGI↩ N_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. 585) object to start handling the alarm signal. Applications must call **setInterval()** (p. 587) before invoking the BEGIN_↩ WATCHDOG_BLOCK() macro.

The END_WATCHDOG_BLOCK() macro disables the watchdog timer, but doesn't affect the assigned interval value. Applications can set the interval once and use the block macros repeatedly. Failure to call **setInterval()** (p. 587) 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. 585) 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_B↩
LOCK() macro or ABORT_WATCHDOG_BLOCK() macro. Failure to do so may result in undefined behavior as a running **Watchdog** (p. 585) timer may expire, forcing a jump into an incompletely initialized function.

Note

Process (p. 148) 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. 585) class. Therefore, applications should not call sleep(3) inside the **Watchdog** (p. 585) block; behavior is undefined in that case, but usually results in cancellation of the **Watchdog** (p. 585) timer.
The **setCanSigJump()** (p. 587), **clearCanSigJump()** (p. 587), **setExpired()** (p. 587) and **clearExpired()** (p. 587) methods are not meant to be used directly by applications, which should use the BEGIN_WA↩
TCHDOG_BLOCK()/END_WATCHDOG_BLOCK() macro pair.

See also

Error::SignalManager (p. 539)

G.147.2 Constructor & Destructor Documentation

G.147.2.1 Watchdog()

```
BiometricEvaluation::Time::Watchdog::Watchdog (
            const uint8_t type )
```

Construct a new **Watchdog** (p. 585) object.

Parameters

in	*type*	The type of timer, ProcessTime or RealTime.

Exceptions

Error::NotImplemented (p. *452*)	The type of watchdog requested is not implemented.
Error::ParameterError (p. *470*)	The type is invalid.

Warning

Watchdog::PROCESSTIME (p. 588) is not supported under Cygwin.

G.147.3 Member Function Documentation

G.147.3.1 clearCanSigJump()

```
void BiometricEvaluation::Time::Watchdog::clearCanSigJump ( )
```

Clears the flag for the **Watchdog** (p. 585) object to indicate that the signal jump block is no longer valid.

G.147.3.2 clearExpired()

```
void BiometricEvaluation::Time::Watchdog::clearExpired ( )
```

Clear the flag indicating the timer expired.

G.147.3.3 expired()

```
bool BiometricEvaluation::Time::Watchdog::expired ( )
```

Indicate whether the watchdog timer expired.

Returns

true if the timer expired, false otherwise.

G.147.3.4 setCanSigJump()

```
void BiometricEvaluation::Time::Watchdog::setCanSigJump ( )
```

Indicate that the signal handler can jump into the application code after handling the signal.

G.147.3.5 setExpired()

```
void BiometricEvaluation::Time::Watchdog::setExpired ( )
```

Set a flag to indicate the timer expired.

G.147.3.6 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. 571) 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.

G.147.3.7 start()

`void BiometricEvaluation::Time::Watchdog::start ( )`

Start a watchdog timer.

Exceptions

***Error::StrategyError** (p. 560)*	Could not register the signal handler, or could not create the timer.

G.147.3.8 stop()

`void BiometricEvaluation::Time::Watchdog::stop ( )`

Stop a watchdog timer.

Exceptions

***Error::StrategyError** (p. 560)*	Could not clear the timer.

G.147.4 Member Data Documentation

G.147.4.1 PROCESSTIME

`const uint8_t BiometricEvaluation::Time::Watchdog::PROCESSTIME = 0  [static]`

A **Watchdog** (p. 585) based on process time.

G.147.4.2 REALTIME

`const uint8_t BiometricEvaluation::Time::Watchdog::REALTIME = 1  [static]`

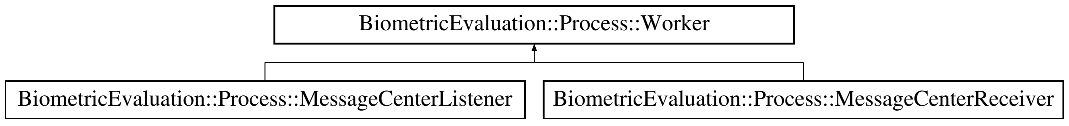
A **Watchdog** (p. 585) based on real (wall clock) time.

G.148 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:


BiometricEvaluation::Process::Worker
BiometricEvaluation::Process::MessageCenterListener
BiometricEvaluation::Process::MessageCenterReceiver

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. 588).*
- double **getParameterAsDouble** (const std::string &name)
  *Obtain a parameter passed to this **Worker** (p. 588) as a double.*
- int64_t **getParameterAsInteger** (const std::string &name)
  *Obtain a parameter passed to this **Worker** (p. 588) as an integer.*
- std::string **getParameterAsString** (const std::string &name)
  *Obtain a parameter passed to this **Worker** (p. 588) as a string.*
- void **setParameter** (const std::string &name, std::shared_ptr< void > argument)
  *Pass a parameter to this **Worker** (p. 588).*
- virtual void **stop** () final
  *Tell this **Worker** (p. 588) to return ASAP.*
- void **closeWorkerPipeEnds** ()
  *Perform initialization for communication from **Worker** (p. 588) to **Manager** (p. 426).*
- void **closeManagerPipeEnds** ()
  *Perform initialization for communication from **Manager** (p. 426) to **Worker** (p. 588).*
- int **getSendingPipe** () const
  *Obtain the pipe used to send messages to this **Worker** (p. 588).*
- int **getReceivingPipe** () const
  *Obtain the pipe used to receive messages to this **Worker** (p. 588).*
- void **sendMessageToManager** (const **Memory::uint8Array** &message)
  *Send a message to the **Manager** (p. 426).*
- void **receiveMessageFromManager** (**Memory::uint8Array** &message)
  *Receive a message from the **Manager** (p. 426).*
- void **_initCommunication** ()
  Perform general communication initialization from Constructor.
- virtual ~**Worker** ()
  ***Worker** (p. 588) destructor.*

Protected Member Functions

- **Worker** ()
  ***Worker** (p. 588) 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. 426).*

G.148.1 Detailed Description

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

G.148.2 Member Function Documentation

G.148.2.1 _initCommunication()

```
void BiometricEvaluation::Process::Worker::_initCommunication ( )
```

Perform general communication initialization from Constructor.

Exceptions

Error::StrategyError *(p. 560)*	**Error** (p. 106) in initialization.

G.148.2.2 closeManagerPipeEnds()

```
void BiometricEvaluation::Process::Worker::closeManagerPipeEnds ( )
```

Perform initialization for communication from **Manager** (p. 426) to **Worker** (p. 588).

Note

Behavior is undefined if called by a non-Worker.

Exceptions

Error::StrategyError *(p. 560)*	Communications not enabled.

G.148.2.3 closeWorkerPipeEnds()

```
void BiometricEvaluation::Process::Worker::closeWorkerPipeEnds ( )
```

Perform initialization for communication from **Worker** (p. 588) to **Manager** (p. 426).

Note

Behavior is undefined if called by a non-Manager.

Exceptions

Error::StrategyError *(p. 560)*	Communications not enabled.

G.148.2.4 getParameter()

```
std::shared_ptr<void> BiometricEvaluation::Process::Worker::getParameter (
             const std::string & name )
```

Obtain a parameter passed to this **Worker** (p. 588).

Parameters

name	The parameter name to retrieve.

Returns

shared_ptr to the parameter argument.

Exceptions

std::out_of_range	name was not set.

G.148.2.5 getParameterAsDouble()

```
double BiometricEvaluation::Process::Worker::getParameterAsDouble (
            const std::string & name )
```

Obtain a parameter passed to this **Worker** (p. 588) as a double.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as a double.

Exceptions

std::out_of_range	name was not set.

G.148.2.6 getParameterAsInteger()

```
int64_t BiometricEvaluation::Process::Worker::getParameterAsInteger (
            const std::string & name )
```

Obtain a parameter passed to this **Worker** (p. 588) as an integer.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as an integer.

Exceptions

std::out_of_range	name was not set.

G.148.2.7 getParameterAsString()

```
std::string BiometricEvaluation::Process::Worker::getParameterAsString (
            const std::string & name )
```

Obtain a parameter passed to this **Worker** (p. 588) as a string.

Parameters

name	The parameter name to retrieve.

Returns

Parameter as a string.

Exceptions

std::out_of_range	name was not set.

G.148.2.8 getReceivingPipe()

```
int BiometricEvaluation::Process::Worker::getReceivingPipe ( ) const
```

Obtain the pipe used to receive messages to this **Worker** (p. 588).

Returns

Receiving pipe.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	**Worker** (p. 588) exiting soon, communication disabled.
Error::StrategyError *(p. 560)*	Communications not enabled.

G.148.2.9 getSendingPipe()

```
int BiometricEvaluation::Process::Worker::getSendingPipe ( ) const
```

Obtain the pipe used to send messages to this **Worker** (p. 588).

Returns

Sending pipe.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	**Worker** (p. 588) exiting soon, communication disabled.
Error::StrategyError *(p. 560)*	Communications not enabled.

G.148.2.10 receiveMessageFromManager()

```
void BiometricEvaluation::Process::Worker::receiveMessageFromManager (
                Memory::uint8Array & message )
```

Receive a message from the **Manager** (p. 426).

Parameters

out	*message*	Buffer to store the received message.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	Widowed pipe.
Error::StrategyError *(p. 560)*	Communications not enabled.

See also

waitForMessage (p. 594)

G.148.2.11 sendMessageToManager()

```
void BiometricEvaluation::Process::Worker::sendMessageToManager (
               const  Memory::uint8Array & message )
```

Send a message to the **Manager** (p. 426).

Parameters

in	*message*	Message to send.

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	Widowed pipe.
Error::StrategyError *(p. 560)*	Communications not enabled.

G.148.2.12 setParameter()

```
void BiometricEvaluation::Process::Worker::setParameter (
               const std::string & name,
               std::shared_ptr< void > argument )
```

Pass a parameter to this **Worker** (p. 588).

Parameters

name	A unique identifier for this parameter
argument	A shared_ptr to the object to store.

G.148.2.13 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.

G.148.2.14 waitForMessage()

```
bool BiometricEvaluation::Process::Worker::waitForMessage (
            int numSeconds = -1 ) const [protected]
```

Block while waiting for a message from the **Manager** (p. 426).

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.

G.148.2.15 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. 331) object, the implementation of **Process::Worker::workerMain()** (p. 594) 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_FAI↩ LURE. The type and contents of the exception is not maintained.

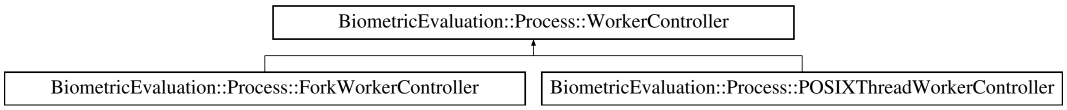
Implemented in **BiometricEvaluation::Process::MessageCenterReceiver** (p. 437), and **Biometric↩ Evaluation::Process::MessageCenterListener** (p. 435).

G.149 BiometricEvaluation::Process::WorkerController Class Reference

Wrapper of a **Worker** (p. 588) returned from a **Process::Manager** (p. 426).

`#include <be_process_workercontroller.h>`

Inheritance diagram for BiometricEvaluation::Process::WorkerController:


BiometricEvaluation::Process::WorkerController
BiometricEvaluation::Process::ForkWorkerController
BiometricEvaluation::Process::POSIXThreadWorkerController

Public Member Functions

- **WorkerController** (std::shared_ptr< **Worker** > worker)
- virtual void **sendMessageToWorker** (const **Memory::uint8Array** &message)

  *Send a message to the **Worker** (p. 588) contained within this **WorkerController** (p. 595).*
- virtual void **setParameter** (const std::string &name, std::shared_ptr< void > argument)

  *Set the parameter to be passed to the **Worker** (p. 588).*
- virtual void **setParameterFromDouble** (const std::string &name, double argument)

  *Set a double parameter to be passed to the **Worker** (p. 588).*
- virtual void **setParameterFromInteger** (const std::string &name, int64_t argument)

  *Set an integer parameter to be passed to the **Worker** (p. 588).*
- virtual void **setParameterFromString** (const std::string &name, const std::string &argument)

  *Set a string parameter to be passed to the **Worker** (p. 588).*
- virtual void **reset** ()

  *Reuse the **Worker** (p. 588).*
- virtual bool **isWorking** () const =0

  *Obtain whether or not **Worker** (p. 588) is working.*
- virtual bool **everWorked** () const =0

  *Obtain whether or not this **Worker** (p. 588) has ever worked.*
- bool **finishedWorking** () const

  *Obtain whether or not this **Worker** (p. 588) has both started and finished its task.*
- std::shared_ptr< **Worker** > **getWorker** () const

  *Obtain the **Worker** (p. 588) instance being wrapped.*
- virtual int32_t **getExitStatus** () const final

  *Obtain the exit status of the wrapped **Worker** (p. 588).*
- virtual ∼**WorkerController** ()

  ***WorkerController** (p. 595) destructor.*

Protected Attributes

- std::shared_ptr< **Worker** > **_worker**
- bool **_rvSet**
- int32_t **_rv**

G.149.1 Detailed Description

Wrapper of a **Worker** (p. 588) returned from a **Process::Manager** (p. 426).

G.149.2 Constructor & Destructor Documentation

G.149.2.1 WorkerController()

```
BiometricEvaluation::Process::WorkerController::WorkerController (
            std::shared_ptr< Worker > worker )
```

WorkerController (p. 595) constructor.

Parameters

worker	The **Worker** (p. 588) instance to wrap.

G.149.3 Member Function Documentation

G.149.3.1 everWorked()

```
virtual bool BiometricEvaluation::Process::WorkerController::everWorked ( ) const  [pure virtual]
```

Obtain whether or not this **Worker** (p. 588) has ever worked.

Returns

true the **Worker** (p. 588) has ever or is currently working, false otherwise.

Note

reset() (p. 597) will change the result of this method.

Implemented in **BiometricEvaluation::Process::ForkWorkerController** (p. 338), and **Biometric↩ Evaluation::Process::POSIXThreadWorkerController** (p. 480).

G.149.3.2 finishedWorking()

```
bool BiometricEvaluation::Process::WorkerController::finishedWorking ( ) const  [inline]
```

Obtain whether or not this **Worker** (p. 588) has both started and finished its task.

Returns

true if the **Worker** (p. 588) has both started and finished performing its task, false otherwise.

Note

reset() (p. 597) will change the result of this method.

G.149.3.3 getExitStatus()

```
virtual int32_t BiometricEvaluation::Process::WorkerController::getExitStatus ( ) const  [final],
[virtual]
```

Obtain the exit status of the wrapped **Worker** (p. 588).

Returns

Exit status of the wrapped **Worker** (p. 588).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	Exit status not set.
Error::StrategyError *(p. 560)*	Exit status not set (e.g., **Worker** (p. 588) has not been started or **Worker** (p. 588) has not finished).

G.149.3.4 getWorker()

```
std::shared_ptr< Worker> BiometricEvaluation::Process::WorkerController::getWorker ( ) const
```

Obtain the **Worker** (p. 588) instance being wrapped.

Returns

Worker (p. 588) instance.

G.149.3.5 isWorking()

```
virtual bool BiometricEvaluation::Process::WorkerController::isWorking ( ) const  [pure virtual]
```

Obtain whether or not **Worker** (p. 588) is working.

Returns

Whether or not the **Worker** (p. 588) is working.

Implemented in **BiometricEvaluation::Process::ForkWorkerController** (p. 339), and **Biometric↩ Evaluation::Process::POSIXThreadWorkerController** (p. 480).

G.149.3.6 reset()

```
virtual void BiometricEvaluation::Process::WorkerController::reset ( )  [virtual]
```

Reuse the **Worker** (p. 588).

Exceptions

Error::ObjectExists *(p. 454)*	The previously started **Worker** (p. 588) is still running.

Reimplemented in **BiometricEvaluation::Process::ForkWorkerController** (p. 339), and **Biometric↩ Evaluation::Process::POSIXThreadWorkerController** (p. 480).

G.149.3.7 sendMessageToWorker()

```
virtual void BiometricEvaluation::Process::WorkerController::sendMessageToWorker (
            const  Memory::uint8Array & message )  [virtual]
```

Send a message to the **Worker** (p. 588) contained within this **WorkerController** (p. 595).

Parameters

message	Message to send to the **Worker** (p. 588).

Exceptions

Error::ObjectDoesNotExist *(p. 453)*	**Worker** (p. 588) receive pipe is closed (**Worker** (p. 588) object likely destroyed).
Error::StrategyError *(p. 560)*	Message sending failed.

G.149.3.8 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. 588).

Parameters

in	*name*	The name representing the argument in the **Worker** (p. 588).
in	*argument*	The argument to be passed to the **Worker** (p. 588).

Note

Subsequent calls to **setParameter()** (p. 598) with the same name will overwrite any exiting argument.

G.149.3.9 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. 588).

Parameters

in	*name*	The name representing the argument in the **Worker** (p. 588).
in	*argument*	The double to be passed to the **Worker** (p. 588).

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

G.149.3.10 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. 588).

Parameters

in	*name*	The name representing the argument in the **Worker** (p. 588).
in	*argument*	The integer to be passed to the **Worker** (p. 588).

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

G.149.3.11 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. 588).

Parameters

in	*name*	The name representing the argument in the **Worker** (p. 588).
in	*argument*	The string to be passed to the **Worker** (p. 588).

Note

Subsequent calls to setParameter∗() with the same name will overwrite any exiting argument.

G.149.4 Member Data Documentation

G.149.4.1 _rv

```
int32_t BiometricEvaluation::Process::WorkerController::_rv  [protected]
```

Exit status from _worker.workerMain()

G.149.4.2 _rvSet

```
bool BiometricEvaluation::Process::WorkerController::_rvSet  [protected]
```

Whether or not _rv contains a true value.

G.149.4.3 _worker

`std::shared_ptr< Worker> BiometricEvaluation::Process::WorkerController::_worker [protected]`

The **Worker** (p. 588) instance that is running in this child

G.150 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.

G.150.1 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.

G.150.2 Constructor & Destructor Documentation

G.150.2.1 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.

G.150.3 Member Function Documentation

G.150.3.1 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.

G.150.3.2 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.

G.150.3.3 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.

G.150.3.4 setNumElements()

```
void BiometricEvaluation::MPI::WorkPackage::setNumElements (
            const uint64_t numElements )
```

Set the number of elements in the package.

Parameters

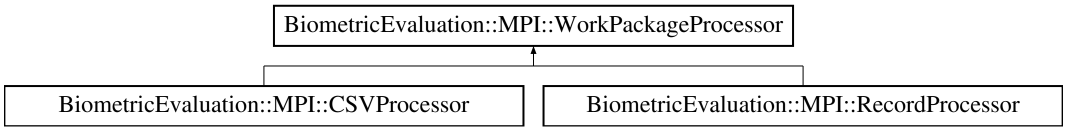
in	*numElements*	The number of appplication-defined elements in the work package.

G.151 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:


BiometricEvaluation::MPI::WorkPackageProcessor
BiometricEvaluation::MPI::CSVProcessor
BiometricEvaluation::MPI::RecordProcessor

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. 148) 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. 416) object that can be used to save message for objects of this class.*
- std::shared_ptr< **IO::Logsheet** > **getLogsheet** ()

  *Obtain the **IO::Logsheet** (p. 416) object that can be used to save message for objects of this class.*

G.151.1 Detailed Description

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

A **WorkPackageProcessor** (p. 602) 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. 602) class is done in the application. Access to the Logsheet object maintained by the framework is provided by this class.

G.151.2 Member Function Documentation

G.151.2.1 getLogsheet()

```
std::shared_ptr< IO::Logsheet> BiometricEvaluation::MPI::WorkPackageProcessor::getLogsheet (
)
```

Obtain the **IO::Logsheet** (p. 416) object that can be used to save message for objects of this class.

Returns

logsheet A shared pointer to the Logsheet object.

G.151.2.2 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. 416) 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. 602). If an error occurs during construction, throw a **Error::Exception** (p. 307) with a message to be caught and logged.

Implemented in **BiometricEvaluation::MPI::CSVProcessor** (p. 287), and **BiometricEvaluation::M↩ PI::RecordProcessor** (p. 498).

G.151.2.3 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. 416) that may be used to save messages generated by the object.

Exceptions

Error::Exception *(p. 307)*	An implementation specific error occurred. The exception string will be logged by the **Framework** (p. 115).

Implemented in **BiometricEvaluation::MPI::CSVProcessor** (p. 288), and **BiometricEvaluation::M↩ PI::RecordProcessor** (p. 498).

G.151.2.4 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. 603) processor objects can be accessed in this method.

Exceptions

Error::Exception *(p. 307)*	An implementation specific error occurred. The exception string will be logged by the **Framework** (p. 115).

G.151.2.5 processWorkPackage()

```
virtual void BiometricEvaluation::MPI::WorkPackageProcessor::processWorkPackage (
            MPI::WorkPackage & workPackage )  [pure virtual]
```

Process (p. 148) 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. 307)*	An fatal error occurred when processing the work package; the processing responsible for this object should shut down.

Implemented in **BiometricEvaluation::MPI::CSVProcessor** (p. 289), and **BiometricEvaluation::M↩ PI::RecordProcessor** (p. 499).

G.151.2.6 setLogsheet()

```
void BiometricEvaluation::MPI::WorkPackageProcessor::setLogsheet (
            std::shared_ptr< IO::Logsheet > & logsheet )
```

Set the **IO::Logsheet** (p. 416) object that can be used to save message for objects of this class.

Parameters

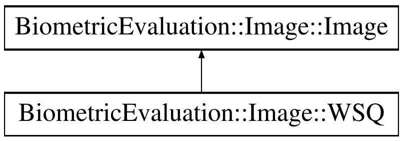
in	*logsheet*	A shared pointer to the Logsheet object.

G.152 BiometricEvaluation::Image::WSQ Class Reference

A WSQ-encoded image.

`#include <be_image_wsq.h>`

Inheritance diagram for BiometricEvaluation::Image::WSQ:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::WSQ

Public Member Functions

- **WSQ** (const uint8_t *data, const uint64_t size)
- **WSQ** (const **Memory::uint8Array** &data)
- **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.

Static Public Member Functions

- static bool **isWSQ** (const uint8_t *data, uint64_t size)

Additional Inherited Members

G.152.1 Detailed Description

A WSQ-encoded image.

G.152.2 Member Function Documentation

G.152.2.1 getRawData()

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

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

AutoArray holding raw image data.

Exceptions

***Error::DataError** (p. 293)*	**Error** (p. 106) decompressing image data.

Implements **BiometricEvaluation::Image::Image** (p. 357).

G.152.2.2 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. 293)*	**Error** (p. 106) decompressing image data.
Error::NotImplemented *(p. 452)*	Unsupported conversion based on source color depth.
Error::ParameterError *(p. 470)*	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. 358).

G.152.2.3 isWSQ()

```
static bool BiometricEvaluation::Image::WSQ::isWSQ (
            const uint8_t * data,
            uint64_t size ) [static]
```

Whether or not data is a **WSQ** (p. 605) image.

Parameters

in	*data*	The buffer to check.
in	*size*	The size of data.

Returns

true if data appears to be a **WSQ** (p. 605) image, false otherwise

G.153 BiometricEvaluation::Feature::Sort::XY Class Reference

```
#include <be_feature_sort.h>
```

Public Member Functions

- bool **operator()** (const **BiometricEvaluation::Feature::MinutiaPoint** &lhs, const **Biometric↩ Evaluation::Feature::MinutiaPoint** &rhs) const

  MinutiaPoint *(p. 440) Cartesian X-Y ascending comparator.*

G.153.1 Detailed Description

Sort (p. 111) by increasing Cartesian X-Y coordinate

G.154 BiometricEvaluation::Feature::Sort::YX Class Reference

```
#include <be_feature_sort.h>
```

Public Member Functions

- bool **operator()** (const **BiometricEvaluation::Feature::MinutiaPoint** &lhs, const **Biometric↩ Evaluation::Feature::MinutiaPoint** &rhs) const

  MinutiaPoint *(p. 440) Cartesian Y-X ascending comparator.*

G.154.1 Detailed Description

Sort (p. 111) by increasing Cartesian Y-X coordinate

Index