Biometric Evaluation Common Framework

Wayne Salamon and Greg Fiumara

Contents

Chapter 1

Introduction

This document describes the framework and application programming interfaces (API) used to support the evaluation of biometric software within the Image Group at NIST. 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 that vendors implement in there software, which is delivered to NIST as a software library. A common test driver is used to call the vendor library to perform the biometric operation. In order to support the common functionality used across all evaluations, such as logging, file input/output, etc., a common framework is used.

Chapter 2

Overview

The Biometric Evaluation Framework (BECommon) is a set of C++[1] 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 appications to constrain the amount of processing time used by a block of code.

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*. All classes within BECommon belong to the top-level *BiometricEvaluation* name space.

Chapter 3

Utility Classes

Chapter 4

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.

4.1 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 informational string stored within that object provides more information on the error.

Listing 5.2 shows an example of exception handling when using the logging classes described in Section 5.3.

4.2 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 4.2 shows application use of the *SignalManager* class.

Listing 4.1: 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.

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

Listing 4.2: Using 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);

        BEGIN_SIGNAL_BLOCK(sigmgr, sigblock2);
        // code that may result in signal generation
        END_SIGNAL_BLOCK(asigmgr, sigblock2);
        if (sigmgr->sigHandled()) {
                cout << "SIGUSR1␣occurred." << endl;
        }
}
```

Chapter 5

Input/Output

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

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

5.1 Utility

The *IO::Utility* class provides static methods that are used to manipulate the file system and other low-level mechanisms. These methods can be used by applications in addition to being used by other classes within the Biometric Evaluation framework.

5.2 Record Management

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

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

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

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

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

Listing 5.1 illustrates the use of a database RecordStore within an application.

Listing 5.1: Using a RecordStore

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

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

    try {
        uint8_t *theData;

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

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

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

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

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

5.3 Logging

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

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

A *LogSheet* is an output stream (subclass of *std::ostringstream*), and therefore can handle built-in types and any class that supports streaming. The example code in 5.2 shows how an application can use a *LogSheet*, contained within a *LogCabinet*, to record operational information.

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

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

The example in Listing 5.2 shows application use of the logging facility.

Listing 5.2: Using a LogSheet within a LogCabinet

```
#include <be_io_logcabinet.h>
```

```
using namespace BiometricEvaluation;
using namespace BiometricEvaluation::IO;

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

5.4 Properties

Listing 5.3: Using a Properties Object

5.5 IO Factory

Chapter 6

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.

6.1 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 6.1 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 6.1: 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;
        }
}
```

6.2 Limiting Execution Time

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

Watch dog timers can be used in conjunction with *SignalManager* in order to both limit the processing time of a call, and handle all signals generated as a result of that call. See 4.2 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 6.2 shows how an application can use a *Watchdog* object to limit the about of process time for a block of code.

Listing 6.2: 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
        BEGIN_WATCHDOG_BLOCK(theDog, watchdogblock1);
                // Do something that may take more than 300 usecs
        END_WATCHDOG_BLOCK(theDog, watchdogblock1);
        if (theDog->expired()) {
                cout << "That␣took␣too␣long." << endl;
                // further processing
        }
{
}
```

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

Chapter 7

Process Information

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

7.1 Process Statistics

When a application is running, there is a need to obtain information of the process executing that application. The Process API can be used by the application itself to gather statistics related to the current amount of memory being used, the number of threads, and other items. Biometric evaluation test drivers are linked against a third party library, and therefore, the application writer does not control the thread count or memory usage for much of the processing. Listing 7.1 shows how an application can use the Statistics API.

Listing 7.1: 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 5.3) contained within a LogCabinet. Applications can start and stop logging at will to this LogSheet. Post-postmortem analysis can then be done on the entries in the LogSheet. Listing 7.2 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 7.2: 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;
}
```

Chapter 8

System

Chapter 9

Image

Bibliography

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

Appendix A

Namespace Index

A.1 Namespace List

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

Appendix B

Class Index

B.1 Class Hierarchy

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

Appendix C

Class Index

C.1 Class List

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

Appendix D

Namespace Documentation

D.1 BiometricEvaluation::Error Namespace Reference

Exceptions, and other error handling.

Classes

- class Exception

  The parent class of all BiometricEvaluation exceptions.

- class FileError

  File error when opening, reading, writing, etc.

- class ParameterError

  An invalid parameter was passed to a constructor or method.

- class ConversionError

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

- class MemoryError

  An error occurred when allocating an object.

- class ObjectExists

  The named object exists and will not be replaced.

- class ObjectDoesNotExist

The named object does not exist.

- class ObjectIsOpen

  The object is already opened.

- class ObjectIsClosed

  The object is closed.

- class StrategyError

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

- class NotImplemented

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

- class SignalManager

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

Functions

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

D.1.1 Detailed Description

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

D.1.2 Function Documentation

D.1.2.1 string BiometricEvaluation::Error::errorStr ( )

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

Returns

The current error message specified by errno.

D.2 BiometricEvaluation::Framework Namespace Reference

Information about the framework.

Functions

- unsigned int getMajorVersion ()

  Framework major version.

- unsigned int getMinorVersion ()

  Framework minor version.

- std::string getCompiler ()

  Compiler used to compile this framework.

- std::string getCompileDate ()

  Date when this framework was compiled.

- std::string getCompileTime ()

  Time when this framework was compiled.

- std::string getCompilerVersion ()

  Version string of compiler used to compile this framework.

D.2.1 Detailed Description

Information about the framework.

D.2.2 Function Documentation

D.2.2.1 unsigned int BiometricEvaluation::Framework::getMajorVersion ( )

Framework major version.

Returns

The major version number of the BiometricFramework

D.2.2.2 unsigned int BiometricEvaluation::Framework::getMinorVersion ( )

Framework minor version.

Returns

The minor version of the BiometricEvaluation framework.

D.2.2.3 std::string BiometricEvaluation::Framework::getCompiler ( )

Compiler used to compile this framework.

Returns

The name of the compiler used to compile this framework.

D.2.2.4 std::string BiometricEvaluation::Framework::getCompileDate ( )

Date when this framework was compiled.

Returns

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

D.2.2.5 std::string BiometricEvaluation::Framework::getCompileTime ( )

Time when this framework was compiled.

Returns

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

D.2.2.6 std::string BiometricEvaluation::Framework::getCompilerVersion ( )

Version string of compiler used to compile this framework.

Returns

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

D.3 BiometricEvaluation::Image Namespace Reference

Classes and methods for manipulating images.

Classes

- class Image

    Represent attributes common to all images.

- class RawImage

    An image with no encoding or compression.

D.3.1 Detailed Description

Classes and methods for manipulating images.

D.4 BiometricEvaluation::IO Namespace Reference

Input/Output functionality.

Namespaces

- namespace Utility

Classes

- struct ManifestEntry
- class ArchiveRecordStore

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

- class DBRecordStore

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

- class Factory
- class FileRecordStore

- class LogSheet

  A class to represent a single logging mechanism.

- class LogCabinet
- class Properties

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

- class RecordStore

  A class to represent a data storage mechanism.

Typedefs

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

D.4.1 Detailed Description

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

D.5 BiometricEvaluation::IO::Utility Namespace Reference

Functions

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

  Create an entire directory tree.

D.5.1 Detailed Description

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

D.5.2 Function Documentation

D.5.2.1 void BiometricEvaluation::IO::Utility::removeDirectory (const string & *directory*, const string & *prefix*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Remove a directory.

Parameters

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

Exceptions

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

D.5.2.2 uint64_t BiometricEvaluation::IO::Utility::getFileSize (const string & *pathname*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Get the size of a file.

Parameters

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

Returns

The file size.

Exceptions

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

D.5.2.3 bool BiometricEvaluation::IO::Utility::fileExists (const string & *pathname*) throw (Error::StrategyError)

Indicate whether a file exists.

Parameters

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

Returns

true if the file exists, false otherwise.

Exceptions

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

D.5.2.4 bool BiometricEvaluation::IO::Utility::validateRootName (const string & *name*)

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

Parameters

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

Returns

true if the name is acceptable, false otherwise.

D.5.2.5 bool BiometricEvaluation::IO::Utility::constructAndCheckPath (const string & *name,* const string & *parentDir,* string & *fullPath*)

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

Parameters

`in`	*name*	The proposed name for the entity; cannot be a pathname.
`in`	*parentDir*	The name of the directory to contain the entity.
`out`	*fullPath*	The complete path to the new entity, when when true is returned; ambiguous when false is returned.

Returns

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

D.5.2.6 int BiometricEvaluation::IO::Utility::makePath (const string & *path,* const mode_t *mode*)

Create an entire directory tree.

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

Parameters

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

Returns

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

D.6 BiometricEvaluation::Memory Namespace Reference

Support for memory-related operations.

Classes

- class AutoBuffer

D.6.1 Detailed Description

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

D.7 BiometricEvaluation::Process Namespace Reference

Process information and controls.

Classes

- class Statistics

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

D.7.1 Detailed Description

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

D.8 BiometricEvaluation::System Namespace Reference

Operating system, hardware, etc.

Functions

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

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

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

  Obtain the amount of real memory in the system.

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

  Obtain the system load average for the last minute.

D.8.1 Detailed Description

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

D.8.2 Function Documentation

D.8.2.1 uint32_t BiometricEvaluation::System::getCPUCount ( ) throw (Error::NotImplemented)

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

Returns

The number of processing units.

Exceptions

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

D.8.2.2 uint64_t BiometricEvaluation::System::getRealMemorySize ( ) throw (Error::NotImplemented)

Obtain the amount of real memory in the system.

Returns

The real memory size, in kilobytes.

Exceptions

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

D.8.2.3 double BiometricEvaluation::System::getLoadAverage ( ) throw (Error::NotImplemented)

Obtain the system load average for the last minute.

Returns

The system load average.

Exceptions

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

D.9 BiometricEvaluation::Text Namespace Reference

Text processing for string objects.

Functions

- void removeLeadingTrailingWhitespace (string &s)

  Remove lead and trailing white space from a string object.

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

  Compute the digest of a string.

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

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

- string filename (const string &path)

  Extract the filename portion of a pathname.

- string dirname (const string &path)

  Extract the directory part of a pathname.

D.9.1 Detailed Description

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

D.9.2 Function Documentation

D.9.2.1 string BiometricEvaluation::Text::digest (const string & *s*, const string & *digest* = `"md5"`) throw (Error::StrategyError)

Compute the digest of a string.

Parameters

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

Returns

An ASCII representation of the hex digits composing the digest.

D.9.2.2 vector<string> BiometricEvaluation::Text::split (const string & *str,* const char *delimiter*)

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

Parameters

`in`	*str*	String to tokenize.
`in`	*delimiter*	Character that defines the end of a token.

Returns

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

Note

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

D.9.2.3 string BiometricEvaluation::Text::filename (const string & *path*)

Extract the filename portion of a pathname.

Parameters

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

Returns

Filename portion of path.

D.9.2.4 string BiometricEvaluation::Text::dirname (const string & *path*)

Extract the directory part of a pathname.

Parameters

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

Returns

Directory portion of path.

D.10 BiometricEvaluation::Time Namespace Reference

Support for time and timers.

Classes

- class Timer

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

- class Watchdog

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

Functions

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

Variables

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

D.10.1 Detailed Description

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

D.11 BiometricEvaluation::Utility Namespace Reference

The Utility package contains helper classes and functions that do not belong in other namespaces.

Classes

- class AutoArray

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

Functions

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

  Compute the digest of a string.

D.11.1 Detailed Description

The Utility package contains helper classes and functions that do not belong in other namespaces.

D.11.2 Function Documentation

D.11.2.1 string BiometricEvaluation::Utility::digest (const void ∗ *buffer,* const size_t *buffer_size,* const string & *digest* = `"md5"`) throw (Error::StrategyError)

Compute the digest of a string.

Parameters

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

Returns

An ASCII representation of the hex digits composing the digest.

Appendix E

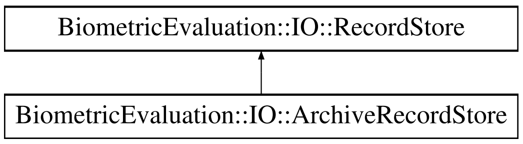
Class Documentation

E.1 BiometricEvaluation::IO::ArchiveRecordStore Class Reference

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

`#include <be_io_archiverecstore.h>`

Inheritance diagram for BiometricEvaluation::IO::ArchiveRecordStore:


BiometricEvaluation::IO::RecordStore
BiometricEvaluation::IO::ArchiveRecordStore

Public Member Functions

- ArchiveRecordStore (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- ArchiveRecordStore (const string &name, const string &parentDir, uint8_t mode=IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- ∼ArchiveRecordStore ()
- uint64_t getSpaceUsed () throw (Error::StrategyError)
- void sync () throw (Error::StrategyError)
- void insert (const string &key, const void ∗const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)

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

Static Public Member Functions

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

E.1.1 Detailed Description

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

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

key offset size

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

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

E.1.2 Constructor & Destructor Documentation

E.1.2.1 BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore (const string & *name*, const string & *description*, const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new ArchiveRecordStore, read/write mode.

Parameters

`in`	*name*	The name of the store.
`in`	*description*	The store's description.
`in`	*parentDir*	The directory where the store is to be created.

Exceptions

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

E.1.2.2 BiometricEvaluation::IO::ArchiveRecordStore::ArchiveRecordStore (const string & *name*, const string & *parentDir*, uint8_t *mode* = `IO::READWRITE`) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing ArchiveRecordStore.

Parameters

`in`	*name*	The name of the store.
`in`	*parentDir*	The directory where the store is to be created.
`in`	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotE	The store does not exist.
Error::StrategyError	An error occurred when accessing the underlying file system.

E.1.2.3 BiometricEvaluation::IO::ArchiveRecordStore::∼ArchiveRecordStore ( )

Destructor.

E.1.3 Member Function Documentation

E.1.3.1 uint64_t BiometricEvaluation::IO::ArchiveRecordStore::getSpaceUsed ( ) throw (Error::StrategyError) [virtual]

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.1.3.2 void BiometricEvaluation::IO::ArchiveRecordStore::sync ( ) throw (Error::StrategyError) [virtual]

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.1.3.3 void BiometricEvaluation::IO::ArchiveRecordStore::insert (const string & *key*, const void ∗const *data*, const uint64_t *size*) throw (Error::ObjectExists, Error::StrategyError) [virtual]

Insert a record into the store.

Parameters

key[in]	The key of the record to be flushed.
data[in]	The data for the record.
size[in]	The size, in bytes, of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.1.3.4 void BiometricEvaluation::IO::ArchiveRecordStore::remove (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Remove a record from the store.

Parameters

`in`	*key*	The key of the record to be removed.

Exceptions

Er-ror::ObjectDoesNotE	A record for the key does not exist.
Er-ror::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

E.1.3.5 uint64_t BiometricEvaluation::IO::ArchiveRecordStore::read (const string & *key*, void ∗const *data*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

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

Parameters

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

Returns

The size of the record.

Exceptions

Er-ror::ObjectDoesNotE	A record for the key does not exist.

Er-ror::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

E.1.3.6 void BiometricEvaluation::IO::ArchiveRecordStore::replace (const string & *key*, const void ∗const *data*, const uint64_t *size*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Replace a complete record in a store.

Parameters

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

Exceptions

Er-ror::ObjectDoesNotE	A record for the key does not exist.
Er-ror::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

E.1.3.7 uint64_t BiometricEvaluation::IO::ArchiveRecordStore::length (const string & *key*) throw (Error::ObjectDoesNotExist) `[virtual]`

Return the length of a record.

Parameters

`in`	*key*	The key of the record.

Returns

The record length.

Exceptions

Er-ror::ObjectDoesNotE	A record for the key does not exist.
Er-ror::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

E.1.3.8 void BiometricEvaluation::IO::ArchiveRecordStore::flush (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.1.3.9 void BiometricEvaluation::IO::ArchiveRecordStore::setCursorAtKey (string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

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

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.1.3.10 void BiometricEvaluation::IO::ArchiveRecordStore::changeName (const string & *name*) throw (Error::ObjectExists, Error::StrategyError) `[virtual]`

Change the name of the RecordStore.

Parameters

name[in]	The new name for the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.1.3.11 bool BiometricEvaluation::IO::ArchiveRecordStore::needsVacuum ( )

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

Returns

true if vacuum() would be beneficial false otherwise

E.1.3.12 static bool BiometricEvaluation::IO::ArchiveRecordStore::needsVacuum (const string & *name*, const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[static]`

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

Parameters

`in`	*name*	The name of the existing RecordStore.
`in`	*parentDir*	Where, in the filesystem, the store is rooted.

Exceptions

Error::ObjectDoesNotE	A record with the given key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Returns

true if vacuum() would be beneficial false otherwise

E.1.3.13 static void BiometricEvaluation::IO::ArchiveRecordStore::vacuum (const string & *name*, const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError) [static]

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

Parameters

in	*name*	The name of the existing RecordStore.
in	*parentDir*	Where, in the file system, the store is rooted.

Exceptions

Error::ObjectDoesNotE	A record with the given key does not exist.
Error::StrategyError	An error occurred when using the underlying storage system.

Note

This is an expensive operation.

E.1.3.14 string BiometricEvaluation::IO::ArchiveRecordStore::getArchiveName ( ) const

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

Returns

Path to archive file.

E.1.3.15 string BiometricEvaluation::IO::ArchiveRecordStore::getManifestName ( ) const

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

Returns

Path to manifest file.

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

- be_io_archiverecstore.h

E.2 BiometricEvaluation::Utility::AutoArray< T > Class Template Reference

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

`#include <be_utility_autoarray.h>`

Public Types

- typedef T value_type

  Convenience typedef for the templated type.

- typedef T * iterator

  Convenience typedef for a pointer to the templated type.

- typedef const T * const_iterator

  Convenience typedef for a pointer to a const templated type.

- typedef T & reference

  Convenience typedef for a reference to the templated type.

- typedef const T & const_reference

  Convenience typedef for a reference to a const templated type.

Public Member Functions

- operator T * ()

  Dereference operator overload.

- reference operator[ ] (ptrdiff_t i)

  Indexing operator overload.

- const_reference operator[ ] (ptrdiff_t i) const

  Const indexing operator overload.

- AutoArray & operator= (const AutoArray &other)

    Assignment operator overload performing a deep copy.

- iterator begin ()

    Obtain an iterator to the beginning of the AutoArray.

- const_iterator begin () const

    Obtain an iterator to the beginning of the AutoArray.

- iterator end ()

    Obtain an iterator to the end of the AutoArray.

- const_iterator end () const

    Obtain an iterator to the end of the AutoArray.

- size_t size () const

    Obtain the number of elements allocated for this AutoArray.

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

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

- AutoArray ()

    Construct an AutoArray.

- AutoArray (size_t size)

    Construct an AutoArray.

- AutoArray (const AutoArray &copy)

    Construct an AutoArray.

E.2.1 Detailed Description

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

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

E.2.2 Constructor & Destructor Documentation

E.2.2.1 template<class T > BiometricEvaluation::Utility::AutoArray< T >::AutoArray ( )

Construct an AutoArray.

The AutoArray will be of size 0.

E.2.2.2 template<class T > BiometricEvaluation::Utility::AutoArray< T >::AutoArray (size_t *size*)

Construct an AutoArray.

Parameters

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

E.2.2.3 template<class T > BiometricEvaluation::Utility::AutoArray< T >::AutoArray (const AutoArray< T > & *copy*)

Construct an AutoArray.

Parameters

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

E.2.3 Member Function Documentation

E.2.3.1 template<class T > BiometricEvaluation::Utility::AutoArray< T >::operator T ∗( )

Dereference operator overload.

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

E.2.3.2 template<class T > BiometricEvaluation::Utility::AutoArray< T >::reference BiometricEvaluation::Utility::AutoArray< T >::operator[ ] (ptrdiff_t *i*)

Indexing operator overload.

Parameters

in	*i*	Index

Returns

Reference to element at index i.

E.2.3.3 template<class T > BiometricEvaluation::Utility::AutoArray< T >::const_reference BiometricEvaluation::Utility::AutoArray< T >::operator[ ] (ptrdiff_t *i*) const

Const indexing operator overload.

Parameters

in	*i*	Index

Returns

Reference to const element at index i.

E.2.3.4 template<class T > BiometricEvaluation::Utility::AutoArray< T > & BiometricEvaluation::Utility::AutoArray< T >::operator= (const AutoArray< T > & *other*)

Assignment operator overload performing a deep copy.

Parameters

in	*other*	AutoArray to be copied

Returns

Reference to a new AutoArray object.

E.2.3.5 template<class T > BiometricEvaluation::Utility::AutoArray< T >::iterator BiometricEvaluation::Utility::AutoArray< T >::begin ( )

Obtain an iterator to the beginning of the AutoArray.

Returns

Pointer to the first element of the AutoArray.

E.2.3.6 template< class T > BiometricEvaluation::Utility::AutoArray< T >::const_iterator BiometricEvaluation::Utility::AutoArray< T >::begin ( ) const

Obtain an iterator to the beginning of the AutoArray.

Returns

Pointer to the const first element of the AutoArray.

E.2.3.7 template< class T > BiometricEvaluation::Utility::AutoArray< T >::iterator BiometricEvaluation::Utility::AutoArray< T >::end ( )

Obtain an iterator to the end of the AutoArray.

Returns

Pointer to the const last element of the AutoArray.

E.2.3.8 template< class T > BiometricEvaluation::Utility::AutoArray< T >::const_iterator BiometricEvaluation::Utility::AutoArray< T >::end ( ) const

Obtain an iterator to the end of the AutoArray.

Returns

Pointer to the const last element of the AutoArray.

E.2.3.9 template< class T > size_t BiometricEvaluation::Utility::AutoArray< T >::size ( ) const

Obtain the number of elements allocated for this AutoArray.

Returns

Number of allocated elements.

E.2.3.10 template<class T > void BiometricEvaluation::Utility::AutoArray< T >::resize (size_t *new_size*, bool *free* = `false`) throw (Error::StrategyError)

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

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

Parameters

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

Exceptions

Error::StrategyError	Problem allocating memory.

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

- be_utility_autoarray.h

E.3 BiometricEvaluation::Memory::AutoBuffer< T > Class Template Reference

Public Types

- typedef T value_type

  Manage a memory buffer.

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

Public Member Functions

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

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

E.3.1 Member Typedef Documentation

E.3.1.1 template<class T> typedef T BiometricEvaluation::Memory::AutoBuffer< T >::value_type

Manage a memory buffer.

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

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

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

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

write_fmttext(FILE *, ANSI_NIST *)

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

int size = obj->num_bytes;

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

- be_memory_autobuffer.h

E.4 be_workorder Struct Reference

Public Attributes

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

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

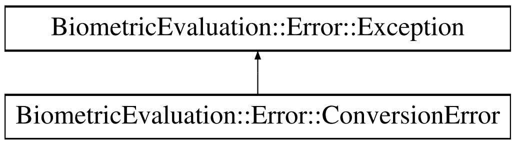
- be_netsdk.h

E.5 BiometricEvaluation::Error::ConversionError Class Reference

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

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

Inheritance diagram for BiometricEvaluation::Error::ConversionError:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ConversionError

Public Member Functions

- ConversionError ()
- ConversionError (string info)

E.5.1 Detailed Description

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

E.5.2 Constructor & Destructor Documentation

E.5.2.1 BiometricEvaluation::Error::ConversionError::ConversionError ( )

Construct a ConversionError object with the default information string.

Returns

The ConversionError object.

E.5.2.2 BiometricEvaluation::Error::ConversionError::ConversionError (string *info*)

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

Returns

The ConversionError object.

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

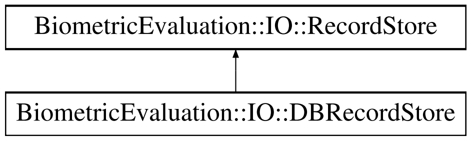
- be_error_exception.h

E.6 BiometricEvaluation::IO::DBRecordStore Class Reference

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

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

Inheritance diagram for BiometricEvaluation::IO::DBRecordStore:


BiometricEvaluation::IO::RecordStore
BiometricEvaluation::IO::DBRecordStore

Public Member Functions

- DBRecordStore (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- DBRecordStore (const string &name, const string &parentDir, uint8_t mode=IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t getSpaceUsed () throw (Error::StrategyError)
- void sync () throw (Error::StrategyError)
- void insert (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
- void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t read (const string &key, void *const data) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void replace (const string &key, const void *const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t length (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void flush (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)

- uint64_t **sequence** (string &key, void ∗const data, int cursor=BE_RECSTORE_-SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)

E.6.1 Detailed Description

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

E.6.2 Constructor & Destructor Documentation

E.6.2.1 BiometricEvaluation::IO::DBRecordStore::DBRecordStore (const string & *name*, const string & *description*, const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new DBRecordStore, read/write mode.

Parameters

in	*name*	The name of the store.
in	*description*	The store's description.
in	*parentDir*	The directory where the store is to be created.

Exceptions

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

E.6.2.2 BiometricEvaluation::IO::DBRecordStore::DBRecordStore (const string & *name*, const string & *parentDir*, uint8_t *mode* = `IO::READWRITE`) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing DBRecordStore.

Parameters

in	*name*	The name of the store.
in	*parentDir*	The directory where the store is to be created.
in	*mode*	Open mode, read-only or read-write.

Exceptions

Er-ror::ObjectDoesNotE	The store does not exist.
Er-ror::StrategyError	An error occurred when accessing the underlying file system.

E.6.3 Member Function Documentation

E.6.3.1 uint64_t BiometricEvaluation::IO::DBRecordStore::getSpaceUsed ( ) throw (Error::StrategyError) `[virtual]`

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

Er-ror::StrategyError	An error occurred when using the underlying storage system.

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.6.3.2 void BiometricEvaluation::IO::DBRecordStore::sync ( ) throw (Error::StrategyError) `[virtual]`

Synchronize the entire record store to persistent storage.

Exceptions

Er-ror::StrategyError	An error occurred when using the underlying storage system.

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.6.3.3 void BiometricEvaluation::IO::DBRecordStore::insert (const string & *key*, const void ∗const *data*, const uint64_t *size*) throw (Error::ObjectExists, Error::StrategyError) `[virtual]`

Insert a record into the store.

Parameters

key[in]	The key of the record to be flushed.
data[in]	The data for the record.
size[in]	The size, in bytes, of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.6.3.4 void BiometricEvaluation::IO::DBRecordStore::remove (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Remove a record from the store.

Parameters

`in`	*key*	The key of the record to be removed.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.6.3.5 uint64_t BiometricEvaluation::IO::DBRecordStore::read (const string & *key*, void ∗const *data*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

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

Parameters

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

Returns

The size of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.6.3.6 void BiometricEvaluation::IO::DBRecordStore::replace (const string & *key*, const void ∗const *data*, const uint64_t *size*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Replace a complete record in a store.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.6.3.7 uint64_t BiometricEvaluation::IO::DBRecordStore::length (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.6.3.8 void BiometricEvaluation::IO::DBRecordStore::flush (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.6.3.9 void BiometricEvaluation::IO::DBRecordStore::setCursorAtKey (string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

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

Parameters

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

Exceptions

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

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

Implements BiometricEvaluation::IO::RecordStore.

E.6.3.10 void BiometricEvaluation::IO::DBRecordStore::changeName (const string & *name*) throw (Error::ObjectExists, Error::StrategyError) `[virtual]`

Change the name of the RecordStore.

Parameters

name[in]	The new name for the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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

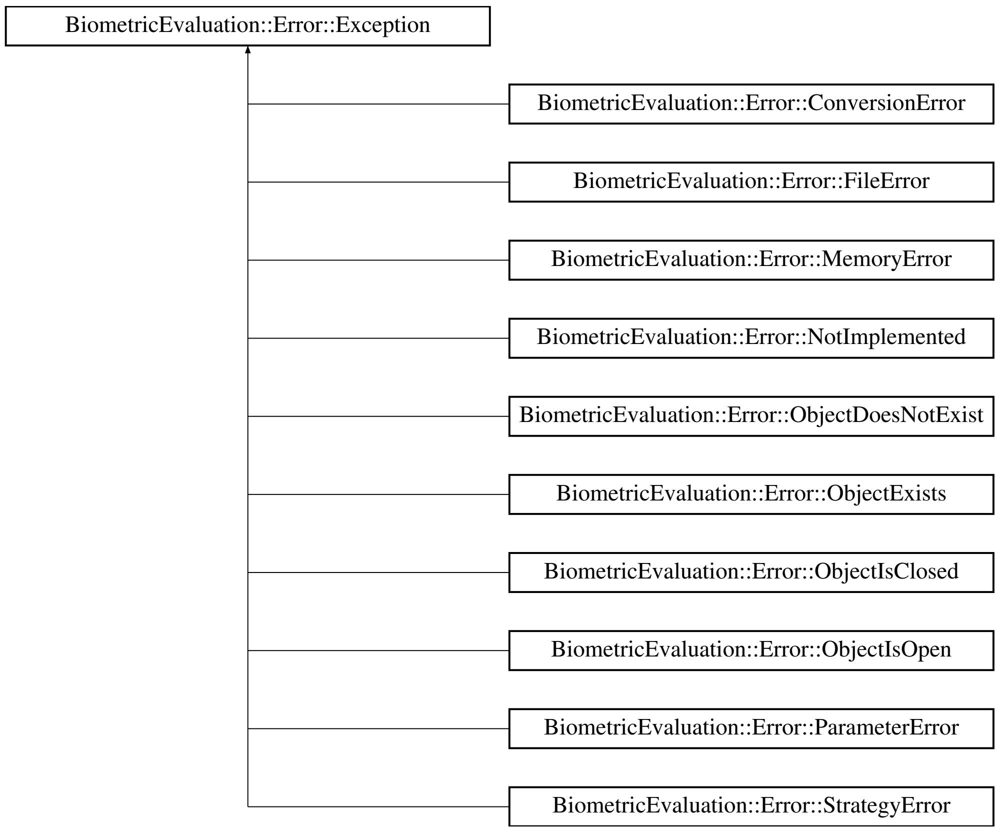
- be_io_dbrecstore.h

E.7 BiometricEvaluation::Error::Exception Class Reference

The parent class of all BiometricEvaluation exceptions.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::Exception:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ConversionError
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

Public Member Functions

- Exception ()
- Exception (string info)
- string getInfo ()

E.7.1 Detailed Description

The parent class of all BiometricEvaluation exceptions. The classes derived from this class will have a default information string set indicating the type of exception. Any additional information string is appended to that string.

E.7.2 Constructor & Destructor Documentation

E.7.2.1 BiometricEvaluation::Error::Exception::Exception ( )

Construct an Exception object without an information string.

Returns

The Exception object.

E.7.2.2 BiometricEvaluation::Error::Exception::Exception (string *info*)

Construct an Exception object with an information string.

Parameters

`in`	*info*	The information string associated with the exception.

Returns

The Exception object.

E.7.3 Member Function Documentation

E.7.3.1 string BiometricEvaluation::Error::Exception::getInfo ( )

Obtain the information string associated with the exception.

Returns

The information string.

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

- be_error_exception.h

E.8 BiometricEvaluation::IO::Factory Class Reference

```
#include <be_io_factory.h>
```

Static Public Member Functions

- static tr1::shared_ptr< RecordStore > openRecordStore (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Open an existing RecordStore and return a managed pointer to the the object representing that store.

- static tr1::shared_ptr< RecordStore > createRecordStore (const string &name, const string &description, const string &type, const string &destDir) throw (Error::ObjectExists, Error::StrategyError)

  Create a new RecordStore and return a managed pointer to the the object representing that store.

E.8.1 Detailed Description

A class to provide constructed objects of classes defined in the BiometricEvaluation::IO package, RecordStores, etc.

E.8.2 Member Function Documentation

E.8.2.1 static tr1::shared_ptr<RecordStore> BiometricEvaluation::IO::Factory::openRecordStore (const string & *name*, const string & *parentDir*, uint8_t *mode* = `READWRITE`) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[static]`

Open an existing RecordStore and return a managed pointer to the the object representing that store.

Applications can open existing record stores without the need to know what type of RecordStore it is.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

`in`	*name*	The name of the store to be opened.
`in`	*parentDir*	Where, in the file system, the store is rooted.
`in`	*mode*	The type of access a client of this RecordStore has.

Returns

An object representing the existing store.

Exceptions

Error::ObjectDoesNotE	The RecordStore does not exist.
Error::StrategyError	An error occurred when using the underlying storage system, or the name is malformed.

E.8.2.2 static tr1::shared_ptr<RecordStore> BiometricEvaluation::IO::Factory::createRecordStore (const string & *name,* const string & *description,* const string & *type,* const string & *destDir*) throw (Error::ObjectExists, Error::StrategyError) `[static]`

Create a new RecordStore and return a managed pointer to the the object representing that store.

The allocated object will be automatically freed when the returned pointer goes out of scope. Applications should not delete the object.

Parameters

`in`	*name*	The name of the store to be created.
`in`	*description*	The description of the store to be created.
`in`	*type*	The type of the store to be created.
`in`	*destDir*	Where, in the file system, the store will be created.

Returns

An auto_ptr to the object representing the created store.

Exceptions

Error::ObjectDoesNotE	The RecordStore does not exist.
Error::StrategyError	An error occurred when using the underlying storage system, or the name is malformed.

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

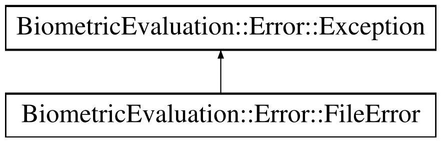
- be_io_factory.h

E.9 BiometricEvaluation::Error::FileError Class Reference

File error when opening, reading, writing, etc.

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

Inheritance diagram for BiometricEvaluation::Error::FileError:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::FileError

Public Member Functions

- FileError ()
- FileError (string info)

E.9.1 Detailed Description

File error when opening, reading, writing, etc.

E.9.2 Constructor & Destructor Documentation

E.9.2.1 BiometricEvaluation::Error::FileError::FileError ( )

Construct a FileError object with the default information string.

Returns

The FileError object.

E.9.2.2 BiometricEvaluation::Error::FileError::FileError (string *info*)

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

Returns

The FileError object.

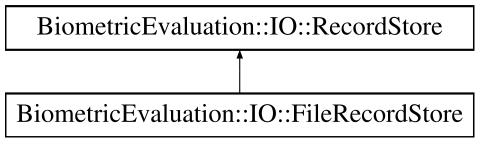
The documentation for this class was generated from the following file:

- be_error_exception.h

E.10 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 string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- FileRecordStore (const string &name, const string &parentDir, uint8_t mode=IO::READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t getSpaceUsed () throw (Error::StrategyError)
- void insert (const string &key, const void ∗const data, const uint64_t size) throw (Error::ObjectExists, Error::StrategyError)
- void remove (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t read (const string &key, void ∗const data) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void replace (const string &key, const void ∗const data, const uint64_t size) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t length (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void flush (const string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- uint64_t **sequence** (string &key, void ∗const data, int cursor=BE_RECSTORE_SEQ_NEXT) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void setCursorAtKey (string &key) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)

Protected Member Functions

- string **canonicalName** (const string &name) const

E.10.1 Detailed Description

Class to represent the record store data storage mechanism implemented as files for each record.

Note

For the methods that take a key parameter, Error::StrategyError will be thrown if the key string is not compliant. A FileRecordStore has the additional requirement that a key name may not contain path delimiter characters ('/' and '\'), or begin with whitespace.

E.10.2 Constructor & Destructor Documentation

E.10.2.1 BiometricEvaluation::IO::FileRecordStore::FileRecordStore (const string & *name*, const string & *description*, const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new FileRecordStore, read/write mode.

Parameters

`in`	*name*	The name of the store.
`in`	*description*	The store's description.
`in`	*parentDir*	The directory where the store is to be created.

Exceptions

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

E.10.2.2 BiometricEvaluation::IO::FileRecordStore::FileRecordStore (const string & *name*, const string & *parentDir*, uint8_t *mode* = `IO::READWRITE`) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing FileRecordStore.

Parameters

`in`	*name*	The name of the store.
`in`	*parentDir*	The directory where the store is to be created.
`in`	*mode*	Open mode, read-only or read-write.

Exceptions

Error::ObjectDoesNotE	The store does not exist.
Error::StrategyError	An error occurred when accessing the underlying file system.

E.10.3 Member Function Documentation

E.10.3.1 uint64_t BiometricEvaluation::IO::FileRecordStore::getSpaceUsed ( ) throw (Error::StrategyError) `[virtual]`

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

E.10.3.2 void BiometricEvaluation::IO::FileRecordStore::insert (const string & *key*, const void ∗const *data*, const uint64_t *size*) throw (Error::ObjectExists, Error::StrategyError) `[virtual]`

Insert a record into the store.

Parameters

key[in]	The key of the record to be flushed.
data[in]	The data for the record.
size[in]	The size, in bytes, of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.10.3.3 void BiometricEvaluation::IO::FileRecordStore::remove (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Remove a record from the store.

Parameters

`in`	*key*	The key of the record to be removed.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.10.3.4 uint64_t BiometricEvaluation::IO::FileRecordStore::read (const string & *key*, void ∗const *data*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

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

Parameters

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

Returns

The size of the record.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.10.3.5 virtual void BiometricEvaluation::IO::FileRecordStore::replace (const string & *key,* const void ∗const *data,* const uint64_t *size*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Replace a complete record in a store.

Parameters

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

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.10.3.6 virtual uint64_t BiometricEvaluation::IO::FileRecordStore::length (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Return the length of a record.

Parameters

in	*key*	The key of the record.

Returns

The record length.

Exceptions

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

Implements BiometricEvaluation::IO::RecordStore.

E.10.3.7 void BiometricEvaluation::IO::FileRecordStore::flush (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

Commit the record's data to storage.

Parameters

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

Exceptions

Er-ror::ObjectDoesNotE	A record for the key does not exist.
Er-ror::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

E.10.3.8 void BiometricEvaluation::IO::FileRecordStore::setCursorAtKey (string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[virtual]`

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

Parameters

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

Exceptions

Er-ror::ObjectDoesNotE	A record for the key does not exist.
Er-ror::StrategyError	An error occurred when using the underlying storage system.

Implements BiometricEvaluation::IO::RecordStore.

E.10.3.9 void BiometricEvaluation::IO::FileRecordStore::changeName (const string & *name*) throw (Error::ObjectExists, Error::StrategyError) `[virtual]`

Change the name of the RecordStore.

Parameters

name[in]	The new name for the RecordStore.

Exceptions

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

Reimplemented from BiometricEvaluation::IO::RecordStore.

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

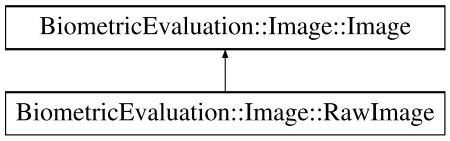
- be_io_filerecstore.h

E.11 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::RawImage

Public Member Functions

- Image (const uint8_t *data, const uint64_t size, const uint64_t width, const uint64_t height, const unsigned int depth, const unsigned int XResolution, const unsigned int YResolution)

    Parent constructor for all Image classes.

- unsigned int getXResolution () const

    Accessor for the X-resolution of the image in terms of pixels per centimeter.

- unsigned int getYResolution () const

    Accessor for the Y-resolution of the image in terms of pixels per centimeter.

- Utility::AutoArray< uint8_t > getData () const

    Accessor for the image data. The data returned is likely encoded in a specialized format.

- virtual Utility::AutoArray< uint8_t > getRawData () const =0

Accessor for the raw image data. The data returned should not be compressed or encoded.

- uint64_t getWidth () const

  Accessor for the width of the image in pixels.

- uint64_t getHeight () const

  Accessor for the height of the image in pixels.

- unsigned int getDepth () const

  Accessor for the color depth of the image in bits.

Protected Attributes

- Utility::AutoArray< uint8_t > **_raw_data**

E.11.1 Detailed Description

Represent attributes common to all images. Images are represented by their size, depth, and resolution on the X and Y axes. The image data can be of any format, raw, JPEG, etc. Implementations of this abstraction provide the getRawData() method to convert image data to 'raw' format.

Image resolution is in pixels per centimeter, and the coordinate system has the origin at the upper left of the image.

E.11.2 Constructor & Destructor Documentation

E.11.2.1 BiometricEvaluation::Image::Image::Image (const uint8_t * *data*, const uint64_t *size*, const uint64_t *width*, const uint64_t *height*, const unsigned int *depth*, const unsigned int *XResolution*, const unsigned int *YResolution*)

Parent constructor for all Image classes.

Parameters

in	*data*	The image data.
in	*size*	The size of the image data, in bytes.
in	*width*	The width of the image, in pixels.
in	*height*	The height of the image, in pixels.
in	*depth*	The image depth, in bits-per-pixel.

in	*XResolution*	The resolution of the image in the horizontal direction, in pixels-per-centimeter.
in	*YResolution*	The resolution of the image in the horizontal direction, in pixels-per-centimeter.

E.11.3 Member Function Documentation

E.11.3.1 unsigned int BiometricEvaluation::Image::Image::getXResolution ( ) const

Accessor for the X-resolution of the image in terms of pixels per centimeter.

Returns

X-resolution (pixel/cm).

E.11.3.2 unsigned int BiometricEvaluation::Image::Image::getYResolution ( ) const

Accessor for the Y-resolution of the image in terms of pixels per centimeter.

Returns

Y-resolution (pixel/cm).

E.11.3.3 Utility::AutoArray<uint8_t> BiometricEvaluation::Image::Image::getData ( ) const

Accessor for the image data. The data returned is likely encoded in a specialized format.

Returns

Image data.

Reimplemented in BiometricEvaluation::Image::RawImage.

E.11.3.4 virtual Utility::AutoArray<uint8_t> BiometricEvaluation::Image::Image::getRawData ( ) const `[pure virtual]`

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

Raw image data.

Implemented in BiometricEvaluation::Image::RawImage.

E.11.3.5 uint64_t BiometricEvaluation::Image::Image::getWidth ( ) const

Accessor for the width of the image in pixels.

Returns

Width of image (pixel).

E.11.3.6 uint64_t BiometricEvaluation::Image::Image::getHeight ( ) const

Accessor for the height of the image in pixels.

Returns

Height of image (pixel).

E.11.3.7 unsigned int BiometricEvaluation::Image::Image::getDepth ( ) const

Accessor for the color depth of the image in bits.

Returns

The color depth of the image (bit).

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

- be_image_image.h

E.12 BiometricEvaluation::IO::LogCabinet Class Reference

```
#include <be_io_logcabinet.h>
```

Public Member Functions

- LogCabinet (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- LogCabinet (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- LogSheet ∗ newLogSheet (const string &name, const string &description) throw (Error::ObjectExists, Error::StrategyError)
- string getName ()
- string getDescription ()
- unsigned int getCount ()

Static Public Member Functions

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

E.12.1 Detailed Description

A class to represent a collection of log sheets.

E.12.2 Constructor & Destructor Documentation

E.12.2.1 BiometricEvaluation::IO::LogCabinet::LogCabinet (const string & *name*, const string & *description*, const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new LogCabinet in the file system.

Parameters

`in`	*name*	The name of the LogCabinet to be created.
`in`	*description*	The text used to describe the cabinet.
`in`	*parentDir*	Where, in the file system, the cabinet is to be stored. This directory must exist.

Returns

An object representing the new log cabinet.

Exceptions

Error::ObjectExists	The cabinet was previously created.

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

E.12.2.2 BiometricEvaluation::IO::LogCabinet::LogCabinet (const string & *name*, const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing LogCabinet.

Parameters

in	*name*	The name of the LogCabinet to be created.
in	*description*	The text used to describe the cabinet.
in	*parentDir*	Where, in the file system, the cabinet is to be stored. This directory must exist.

Returns

An object representing the log cabinet.

Exceptions

Error::ObjectDoesNotE	The cabinet does not exist in the file system.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

E.12.3 Member Function Documentation

E.12.3.1 LogSheet∗ BiometricEvaluation::IO::LogCabinet::newLogSheet (const string & *name*, const string & *description*) throw (Error::ObjectExists, Error::StrategyError)

Create a new LogSheet within the LogCabinet.

Parameters

in	*name*	The name of the LogSheet to be created.
in	*description*	The text used to describe the sheet. This text is written into the log file prior to any entries.

in	*parentDir*	Where, in the file system, the sheet is to be stored. This directory must exist.

Returns

An object pointer to the new log sheet.

Exceptions

Error::ObjectExists	The sheet was previously created.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

E.12.3.2 string BiometricEvaluation::IO::LogCabinet::getName ( )

Obtain the name of the LogCabinet.

@ returns The name of the LogCabinet.

E.12.3.3 string BiometricEvaluation::IO::LogCabinet::getDescription ( )

Obtain the description of the LogCabinet.

@ returns The description of the LogCabinet.

E.12.3.4 unsigned int BiometricEvaluation::IO::LogCabinet::getCount ( )

Obtain the number of items in the LogCabinet.

@ returns The number of LogSheets manages by the cabinet.

E.12.3.5 static void BiometricEvaluation::IO::LogCabinet::remove (const string & *name*, const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[static]`

Remove a LogCabinet.

Parameters

in	*name*	The name of the LogCabinet to be removed.
in	*parentDir*	Where, in the file system, the sheet is to be stored. This directory must exist.

Exceptions

Error::ObjectDoesNotE	The LogCabinet does not exist.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

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

- be_io_logcabinet.h

E.13 BiometricEvaluation::IO::LogSheet Class Reference

A class to represent a single logging mechanism.

```
#include <be_io_logcabinet.h>
```

Public Member Functions

- LogSheet (const string &name, const string &description, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)

  Create a new log sheet.

- LogSheet (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Open an existing new log sheet for appending.

- void write (const string &entry) throw (Error::StrategyError)
- void writeComment (const string &comment) throw (Error::StrategyError)
- void newEntry () throw (Error::StrategyError)
- string getCurrentEntry ()
- void resetCurrentEntry ()
- uint32_t getCurrentEntryNumber ()
- void sync () throw (Error::StrategyError)
- void setAutoSync (bool state)

Static Public Attributes

- static const char CommentDelimiter = ’#’
- static const char EntryDelimiter = ’E’
- static const string DescriptionTag

E.13.1 Detailed Description

A class to represent a single logging mechanism. A LogSheet is a string stream, so applications can write into the stream as a staging area using the $<<$ operator, then start a new entry by calling newEntry(). Entries in the log file are prefixed with an entry number, which is incremented when the entry is written (either by directly calling write(), or calling newEntry()).

A LogSheet object can be constructed and passed back to the client by the LogCabinet object. All sheets created in the manner are placed in a common area maintained by the cabinet.

Note

By default, the entries in the LogSheet may not be immediately written to the file system, depending on the buffering behavior of the operating system. Applications can force a write by invoking sync(), or force a write at every new log entry by invoking setAutoSync(true).
Entries created by applications may be composed of more than one line (each separated by the newline character). The text at the beginning of a line should not "look like" an entry number:
Edddd
i.e. the entry delimiter followed by some digits. LogSheet won't check for that condition, but any existing LogSheet that is re-opened for append may have an incorrect starting entry number.

E.13.2 Constructor & Destructor Documentation

E.13.2.1 BiometricEvaluation::IO::LogSheet::LogSheet (const string & *name*, const string & *description*, const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Create a new log sheet.

Parameters

`in`	*name*	The name of the LogSheet to be created.
`in`	*description*	The text used to describe the sheet. This text is written into the log file prior to any entries.
`in`	*parentDir*	Where, in the file system, the sheet is to be stored. This directory must exist.

Returns

An object representing the new log sheet.

Exceptions

Error::ObjectExists	The sheet was previously created.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

E.13.2.2 BiometricEvaluation::IO::LogSheet::LogSheet (const string & *name*, const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Open an existing new log sheet for appending.

On open, the current entry counter is set to the last entry number plus one.

Note

Opening a large LogSheet may be a costly operation.

Parameters

`in`	*name*	The name of the LogSheet to be opened.
`in`	*parentDir*	Where, in the file system, the sheet is stored.

Returns

An object representing the existing log sheet.

Exceptions

Error::ObjectDoesNotE	The sheet does not exist.
Error::StrategyError	An error occurred when using the underlying file system, or name or parentDir is malformed.

E.13.3 Member Function Documentation

E.13.3.1 void BiometricEvaluation::IO::LogSheet::write (const string & *entry*) throw (Error::StrategyError)

Write a string as an entry to the log file. This does not affect the current log entry buffer, but does increment the entry number.

Parameters

`in`	*entry*	The text of the log entry.

Exceptions

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

E.13.3.2 void BiometricEvaluation::IO::LogSheet::writeComment (const string & *comment*) throw (Error::StrategyError)

Write a string as a comment to the log file. This does not affect the current log entry buffer, and does not increment the entry number. A comment line is prefixed with CommentDelimiter followed by a space by this method.

Parameters

`in`	*comment*	The text of the comment.

Exceptions

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

E.13.3.3 void BiometricEvaluation::IO::LogSheet::newEntry ( ) throw (Error::StrategyError)

Start a new entry, causing the existing entry to be closed. Applications do not have to call this method for the first entry, however, as the stream is ready for writing upon construction.

Exceptions

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

E.13.3.4 string BiometricEvaluation::IO::LogSheet::getCurrentEntry ( )

Obtain the contents of the current entry currently under construction.

Returns

The text of the current entry.

E.13.3.5 void BiometricEvaluation::IO::LogSheet::resetCurrentEntry ( )

Reset the current entry buffer to the beginning.

E.13.3.6 uint32_t BiometricEvaluation::IO::LogSheet::getCurrentEntryNumber ( )

Obtain the current entry number.

Returns

The current entry number.

E.13.3.7 void BiometricEvaluation::IO::LogSheet::sync ( ) throw (Error::StrategyError)

Synchronize any buffered data to the underlying log file. This syncing is dependent on the behavior of the underlying filesystem and operating system.

Exceptions

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

E.13.3.8 void BiometricEvaluation::IO::LogSheet::setAutoSync (bool *state*)

Turn on/off auto-sync of the data. Applications can gain loggin performance by turning off auto-sysnc, or gain reliability by turning it on.

Parameters

state	When true, the data is sync'd whenever newEntry() is or write() is called. When false, sync() must be called to force a write.

E.13.4 Member Data Documentation

E.13.4.1 const char BiometricEvaluation::IO::LogSheet::CommentDelimiter = '#' [static]

The delimiter for a comment line in the log sheet.

E.13.4.2 const char BiometricEvaluation::IO::LogSheet::EntryDelimiter = 'E' `[static]`

The delimiter for an entry line in the log sheet.

E.13.4.3 const string BiometricEvaluation::IO::LogSheet::DescriptionTag `[static]`

The tag for the description string.

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

- be_io_logcabinet.h

E.14 BiometricEvaluation::IO::ManifestEntry Struct Reference

Public Attributes

- long **offset**
- uint64_t **size**

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

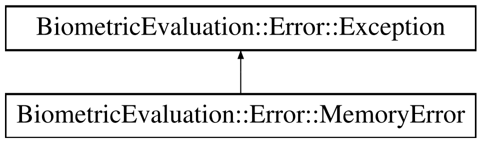
- be_io_archiverecstore.h

E.15 BiometricEvaluation::Error::MemoryError Class Reference

An error occurred when allocating an object.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::MemoryError:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::MemoryError

Public Member Functions

- MemoryError ()
- MemoryError (string info)

E.15.1 Detailed Description

An error occurred when allocating an object.

E.15.2 Constructor & Destructor Documentation

E.15.2.1 BiometricEvaluation::Error::MemoryError::MemoryError ( )

Construct a MemoryError object with the default information string.

Returns

The MemoryError object.

E.15.2.2 BiometricEvaluation::Error::MemoryError::MemoryError (string *info*)

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

Returns

The MemoryError object.

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

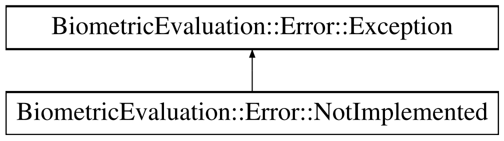
- be_error_exception.h

E.16 BiometricEvaluation::Error::NotImplemented Class Reference

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

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

Inheritance diagram for BiometricEvaluation::Error::NotImplemented:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::NotImplemented

Public Member Functions

- NotImplemented ()
- NotImplemented (string info)

E.16.1 Detailed Description

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

E.16.2 Constructor & Destructor Documentation

E.16.2.1 BiometricEvaluation::Error::NotImplemented::NotImplemented ( )

Construct a NotImplemented object with the default information string.

Returns

The NotImplemented object.

E.16.2.2 BiometricEvaluation::Error::NotImplemented::NotImplemented (string *info*)

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

Returns

The NotImplemented object.

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

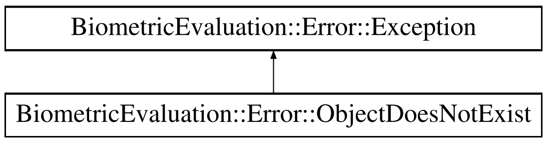
- be_error_exception.h

E.17 BiometricEvaluation::Error::ObjectDoesNotExist Class Reference

The named object does not exist.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectDoesNotExist:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ObjectDoesNotExist

Public Member Functions

- ObjectDoesNotExist ()
- ObjectDoesNotExist (string info)

E.17.1 Detailed Description

The named object does not exist.

E.17.2 Constructor & Destructor Documentation

E.17.2.1 BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist ( )

Construct a ObjectDoesNotExist object with the default information string.

Returns

The ObjectDoesNotExist object.

E.17.2.2 BiometricEvaluation::Error::ObjectDoesNotExist::ObjectDoesNotExist (string *info*)

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

Returns

The ObjectDoesNotExist object.

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

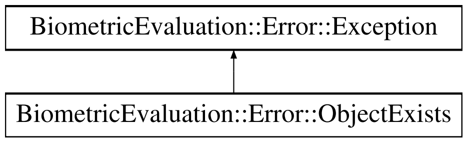
- be_error_exception.h

E.18 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:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ObjectExists

Public Member Functions

- ObjectExists ()
- ObjectExists (string info)

E.18.1 Detailed Description

The named object exists and will not be replaced.

E.18.2 Constructor & Destructor Documentation

E.18.2.1 BiometricEvaluation::Error::ObjectExists::ObjectExists ( )

Construct a ObjectExists object with the default information string.

Returns

The ObjectExists object.

E.18.2.2 BiometricEvaluation::Error::ObjectExists::ObjectExists (string *info*)

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

Returns

The ObjectExists object.

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

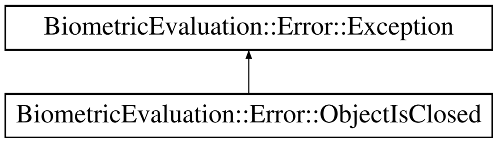
- be_error_exception.h

E.19 BiometricEvaluation::Error::ObjectIsClosed Class Reference

The object is closed.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectIsClosed:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ObjectIsClosed

Public Member Functions

- ObjectIsClosed ()
- ObjectIsClosed (string info)

E.19.1 Detailed Description

The object is closed.

E.19.2 Constructor & Destructor Documentation

E.19.2.1 BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed ( )

Construct a ObjectIsClosed object with the default information string.

Returns

The ObjectIsClosed object.

E.19.2.2 BiometricEvaluation::Error::ObjectIsClosed::ObjectIsClosed (string *info*)

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

Returns

The ObjectIsClosed object.

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

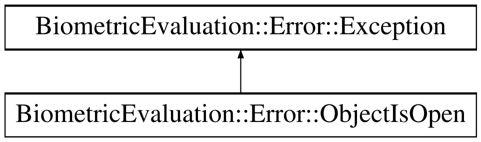
- be_error_exception.h

E.20 BiometricEvaluation::Error::ObjectIsOpen Class Reference

The object is already opened.

`#include <be_error_exception.h>`

Inheritance diagram for BiometricEvaluation::Error::ObjectIsOpen:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ObjectIsOpen

Public Member Functions

- ObjectIsOpen ()
- ObjectIsOpen (string info)

E.20.1 Detailed Description

The object is already opened.

E.20.2 Constructor & Destructor Documentation

E.20.2.1 BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen ( )

Construct a ObjectIsOpen object with the default information string.

Returns

The ObjectIsOpen object.

E.20.2.2 BiometricEvaluation::Error::ObjectIsOpen::ObjectIsOpen (string *info*)

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

Returns

The ObjectIsOpen object.

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

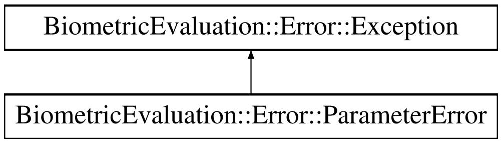
- be_error_exception.h

E.21 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:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::ParameterError

Public Member Functions

- ParameterError ()
- ParameterError (string info)

E.21.1 Detailed Description

An invalid parameter was passed to a constructor or method.

E.21.2 Constructor & Destructor Documentation

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

Construct a ParameterError object with the default information string.

Returns

The ParameterError object.

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

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

Returns

The ParameterError object.

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

- be_error_exception.h

E.22 BiometricEvaluation::IO::Properties Class Reference

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

`#include <be_io_properties.h>`

Public Types

- typedef PropertiesMap::const_iterator **Properties_iter**

Public Member Functions

- Properties (const string &filename, uint8_t mode=IO::READWRITE) throw (Error::StrategyError, Error::FileError)
- void setProperty (const string &property, const string &value) throw (Error::StrategyError)
- void setPropertyFromInteger (const string &property, int64_t value) throw (Error::StrategyError)
- void removeProperty (const string &property) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- string getProperty (const string &property) throw (Error::ObjectDoesNotExist)
- int64_t getPropertyAsInteger (const string &property) throw (Error::ObjectDoesNotExist, Error::ConversionError)
- void sync () throw (Error::FileError, Error::StrategyError)
- void changeName (const string &filename) throw (Error::StrategyError)

E.22.1 Detailed Description

A Properties class is used to maintain key/value pairs of strings, with each property matched to one value. The properties are read from a file that is specified in the constructor, and will be created if it does not exist.

An example file might look like this:

```
 *      Name = John Smith
 *      Age = 32
 *      Favorite Hex Number = 0xffff
 *
```

For property keys and values, leading and trailing whitespace is removed, therefore a the call

```
     props->setProperty("  My property   ", "   A Value  ");
```

results in an entry in the property file as

```
 *      My property = A value
 *
```

Therefore, the property names "Foo", " Foo", "Foo " are equivalent.

E.22.2 Constructor & Destructor Documentation

E.22.2.1 BiometricEvaluation::IO::Properties::Properties (const string & *filename,* uint8_t *mode* = `IO::READWRITE`) throw (Error::StrategyError, Error::FileError)

Construct a new Properties object from an existing or to be created properties file. The constructor will create the file when it does not exist.

Parameters

`in`	*filename*	The name of the file to store the properties. This can be the empty string, meaning the properties are to be stored in memory only.
`in`	*mode*	The read/write mode of the object.

Returns

An object representing the properties set.

Exceptions

Error::StrategyError	A line in the properties file is malformed.
Error::FileError	An error occurred when using the underlying storage system.

E.22.3 Member Function Documentation

E.22.3.1 void BiometricEvaluation::IO::Properties::setProperty (const string & *property,* const string & *value*) throw (Error::StrategyError)

Set a property with a value. Both the property and value will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise, the property will be created.

Parameters

`in`	*property*	The name of the property to set.
`in`	*value*	The value associated with the property.

Exceptions

Error::StrategyError	The Properties object is read-only.

E.22.3.2 void BiometricEvaluation::IO::Properties::setPropertyFromInteger (const string & *property*, int64_t *value*) throw (Error::StrategyError)

Set a property with an integer value. The property will have leading and trailing whitespace removed. If the property already exists in the set, its value will be replaced with the new value; otherwise the property will be created.

Parameters

in	*property*	The name of the property to set.
in	*value*	The value associated with the property.

Exceptions

Error::StrategyError	The Properties object is read-only.

E.22.3.3 void BiometricEvaluation::IO::Properties::removeProperty (const string & *property*) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Remove a property.

Parameters

in	*property*	The name of the property to set.

Exceptions

Error::ObjectDoesNotE	The named property does not exist.
Error::StrategyError	The Properties object is read-only.

E.22.3.4 string BiometricEvaluation::IO::Properties::getProperty (const string & *property*) throw (Error::ObjectDoesNotExist)

Retrieve a property value as a string object.

Parameters

in	*property*	The name of the property to get.

Exceptions

Error::ObjectDoesNotE	The named property does not exist.

E.22.3.5 int64_t BiometricEvaluation::IO::Properties::getPropertyAsInteger (const string & *property*) throw (Error::ObjectDoesNotExist, Error::ConversionError)

Retrieve a property value as an integer value. Integer value strings for properties can represent either decimal or hexadecimal values, which must be preceded with either "0x" or "0X".

Parameters

`in`	*property*	The name of the property to get.

Exceptions

Error::ObjectDoesNotE	The named property does not exist.
Error::ConversionErroi	The property value cannot be converted, usually due to non-numeric characters in the string.

E.22.3.6 void BiometricEvaluation::IO::Properties::sync ( ) throw (Error::FileError, Error::StrategyError)

Write the properties to the underlying file, synchronizing the in-memory and on-disk versions.

Exceptions

Error::FileError	An error occurred when using the underlying storage system.
Error::StrategyError	The object was constructed with NULL as the file name, or is read-only.

E.22.3.7 void BiometricEvaluation::IO::Properties::changeName (const string & *filename*) throw (Error::StrategyError)

Change the name of the Properties, which means changing the name of the underlying file that stores the properties. The empty string ("") can be used to indicate no backing file.

Note

No check is made that the file is writeable at this time.

Parameters

`in`	*filename*	The name of the properties file.

Exceptions

Error::StrategyError	The object is read-only.

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

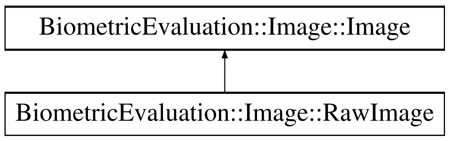
- be_io_properties.h

E.23 BiometricEvaluation::Image::RawImage Class Reference

An image with no encoding or compression.

`#include <be_image_rawimage.h>`

Inheritance diagram for BiometricEvaluation::Image::RawImage:


BiometricEvaluation::Image::Image
BiometricEvaluation::Image::RawImage

Public Member Functions

- RawImage (const uint8_t ∗data, const uint64_t size, const uint64_t width, const uint64_t height, const unsigned int depth, const unsigned int XResolution, const unsigned int YResolution)

  Construct a RawImage object.

- Utility::AutoArray< uint8_t > getData () const

  Accessor for the image data. The data returned is likely encoded in a specialized format.

- Utility::AutoArray< uint8_t > getRawData () const

Accessor for the raw image data. The data returned should not be compressed or encoded.

E.23.1 Detailed Description

An image with no encoding or compression.

E.23.2 Constructor & Destructor Documentation

E.23.2.1 BiometricEvaluation::Image::RawImage::RawImage (const uint8_t * *data*, const uint64_t *size*, const uint64_t *width*, const uint64_t *height*, const unsigned int *depth*, const unsigned int *XResolution*, const unsigned int *YResolution*)

Construct a RawImage object.

Parameters

in	*data*	The image data.
in	*size*	The size of the image data, in bytes.
in	*width*	The width of the image, in pixels.
in	*height*	The height of the image, in pixels.
in	*depth*	The image depth, in bits-per-pixel.
in	*XResolution*	The resolution of the image in the horizontal direction, in pixels-per-centimeter.
in	*YResolution*	The resolution of the image in the horizontal direction, in pixels-per-centimeter.

E.23.3 Member Function Documentation

E.23.3.1 Utility::AutoArray<uint8_t> BiometricEvaluation::Image::RawImage::getData ( ) const

Accessor for the image data. The data returned is likely encoded in a specialized format.

Returns

Image data.

Reimplemented from BiometricEvaluation::Image::Image.

E.23.3.2 Utility::AutoArray<uint8_t> BiometricEvaluation::Image::RawImage::getRawData ( ) const [virtual]

Accessor for the raw image data. The data returned should not be compressed or encoded.

Returns

Raw image data.

Implements BiometricEvaluation::Image::Image.

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

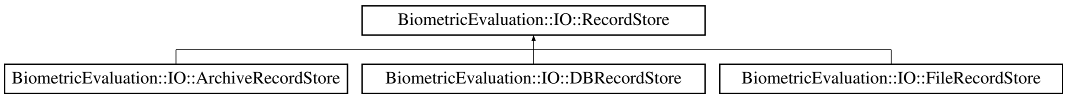
- be_image_rawimage.h

E.24 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::DBRecordStore
BiometricEvaluation::IO::FileRecordStore

Public Member Functions

- RecordStore (const string &name, const string &description, const string &type, const string &parentDir) throw (Error::ObjectExists, Error::StrategyError)
- RecordStore (const string &name, const string &parentDir, uint8_t mode=READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- string getName () const
- string getDescription () const
- unsigned int getCount () const
- virtual void changeName (const string &name) throw (Error::ObjectExists, Error::StrategyError)
- virtual void changeDescription (const string &description) throw (Error::StrategyError)
- virtual uint64_t getSpaceUsed () throw (Error::StrategyError)

- virtual void sync () throw (Error::StrategyError)
- virtual void insert (const string &key, const void ∗const data, const uint64_-t size)=0 throw (Error::ObjectExists, Error::StrategyError)
- virtual void remove (const string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t read (const string &key, void ∗const data)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void replace (const string &key, const void ∗const data, const uint64_t size)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t length (const string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void flush (const string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual uint64_t **sequence** (string &key, void ∗const data=NULL, int cursor=BE_-RECSTORE_SEQ_NEXT)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)
- virtual void setCursorAtKey (string &key)=0 throw (Error::ObjectDoesNotExist, Error::StrategyError)

Static Public Member Functions

- static void removeRecordStore (const string &name, const string &parentDir) throw (Error::ObjectDoesNotExist, Error::StrategyError)
- static void mergeRecordStores (const string &mergedName, const string &mergedDescription, const string &parentDir, const string &type, RecordStore ∗recordStores[ ], size_t numRecordStores) throw (Error::ObjectExists, Error::StrategyError)
- static void mergeRecordStores (const string &mergedName, const string &mergedDescription, const string &parentDir, const string &type, tr1::shared_-ptr< RecordStore > recordStores[ ], size_t numRecordStores) throw (Error::ObjectExists, Error::StrategyError)

Static Public Attributes

- static const string CONTROLFILENAME
- static const string NAMEPROPERTY
- static const string **DESCRIPTIONPROPERTY**
- static const string **COUNTPROPERTY**
- static const string **TYPEPROPERTY**
- static const string BERKELEYDBTYPE
- static const string **ARCHIVETYPE**
- static const string **FILETYPE**
- static const int BE_RECSTORE_SEQ_START = 1
- static const int **BE_RECSTORE_SEQ_NEXT** = 2

Protected Member Functions

- uint8_t **getMode** () const
- string **getDirectory** () const
- string **getParentDirectory** () const
- string **canonicalName** (const string &name) const
- int **getCursor** () const
- void **setCursor** (int cursor)

E.24.1 Detailed Description

A class to represent a data storage mechanism. A RecordStore is an abstraction that associates keys with a specific record. Implementations of this abstraction can store the records in any format supported by the operating system, such as files or databases, rooted in the file system.

See also

IO::ArchiveRecordStore, IO::DBRecordStore, IO::FileRecordStore.

E.24.2 Constructor & Destructor Documentation

E.24.2.1 BiometricEvaluation::IO::RecordStore::RecordStore (const string & *name*, const string & *description*, const string & *type*, const string & *parentDir*) throw (Error::ObjectExists, Error::StrategyError)

Constructor to create a new RecordStore.

Parameters

in	*name*	The name of the RecordStore to be created.
in	*description*	The text used to describe the store.
in	*type*	The type of RecordStore.
in	*parentDir*	Where, in the file system, the store is to be rooted. This directory must exist.

Returns

An object representing the new, empty store.

Exceptions

Error::ObjectExists	The store was previously created, or the directory where it would be created exists.

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

E.24.2.2 BiometricEvaluation::IO::RecordStore::RecordStore (const string & *name*, const string & *parentDir*, uint8_t *mode* = READWRITE) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Constructor to open an existing RecordStore.

Parameters

in	*name*	The name of the store to be opened.
in	*parentDir*	Where, in the file system, the store is rooted.
in	*mode*	The type of access a client of this RecordStore has.

Returns

An object representing the existing store.

Exceptions

Error::ObjectDoesNotE	The RecordStore does not exist.
Error::StrategyError	An error occurred when using the underlying storage system, or the name is malformed.

E.24.3 Member Function Documentation

E.24.3.1 string BiometricEvaluation::IO::RecordStore::getName ( ) const

Return the name of the RecordStore.

Returns

The RecordStore's name.

E.24.3.2 string BiometricEvaluation::IO::RecordStore::getDescription ( ) const

Obtain a textual description of the RecordStore.

Returns

The RecordStore's description.

E.24.3.3 unsigned int BiometricEvaluation::IO::RecordStore::getCount ( ) const

Obtain the number of items in the RecordStore.

Returns

The number of items in the RecordStore.

E.24.3.4 virtual void BiometricEvaluation::IO::RecordStore::changeName (const string & *name*) throw (Error::ObjectExists, Error::StrategyError) `[virtual]`

Change the name of the RecordStore.

Parameters

name[in]	The new name for the RecordStore.

Exceptions

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

Reimplemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.24.3.5 virtual void BiometricEvaluation::IO::RecordStore::changeDescription (const string & *description*) throw (Error::StrategyError) `[virtual]`

Change the description of the RecordStore.

Parameters

`in`	*description*	The new description.

Exceptions

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

E.24.3.6 virtual uint64_t BiometricEvaluation::IO::RecordStore::getSpaceUsed ( ) throw (Error::StrategyError) `[virtual]`

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

Returns

The amount of backing storage used by the RecordStore.

Exceptions

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

Reimplemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.24.3.7 virtual void BiometricEvaluation::IO::RecordStore::sync ( ) throw (Error::StrategyError) `[virtual]`

Synchronize the entire record store to persistent storage.

Exceptions

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

Reimplemented in BiometricEvaluation::IO::ArchiveRecordStore, and BiometricEvaluation::IO::DBRecordStore.

E.24.3.8 virtual void BiometricEvaluation::IO::RecordStore::insert (const string & *key*, const void ∗const *data*, const uint64_t *size*) throw (Error::ObjectExists, Error::StrategyError) `[pure virtual]`

Insert a record into the store.

Parameters

key[in]	The key of the record to be flushed.
data[in]	The data for the record.
size[in]	The size, in bytes, of the record.

Exceptions

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

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.24.3.9 virtual void BiometricEvaluation::IO::RecordStore::remove (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[pure virtual]`

Remove a record from the store.

Parameters

`in`	*key*	The key of the record to be removed.

Exceptions

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

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.24.3.10 virtual uint64_t BiometricEvaluation::IO::RecordStore::read (const string & *key*, void ∗const *data*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[pure virtual]`

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

Parameters

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

Returns

The size of the record.

Exceptions

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

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.24.3.11 virtual void BiometricEvaluation::IO::RecordStore::replace (const string & *key*, const void ∗const *data*, const uint64_t *size*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[pure virtual]`

Replace a complete record in a store.

Parameters

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

Exceptions

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

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.24.3.12 virtual uint64_t BiometricEvaluation::IO::RecordStore::length (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[pure virtual]`

Return the length of a record.

Parameters

`in`	*key*	The key of the record.

Returns

The record length.

Exceptions

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

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.24.3.13 virtual void BiometricEvaluation::IO::RecordStore::flush (const string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[pure virtual]`

Commit the record's data to storage.

Parameters

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

Exceptions

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

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.24.3.14 virtual void BiometricEvaluation::IO::RecordStore::setCursorAtKey (string & *key*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[pure virtual]`

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

Parameters

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

Exceptions

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

Er-ror::StrategyError	An error occurred when using the underlying storage system.

Implemented in BiometricEvaluation::IO::ArchiveRecordStore, BiometricEvaluation::IO::DBRecordStore, and BiometricEvaluation::IO::FileRecordStore.

E.24.3.15 static void BiometricEvaluation::IO::RecordStore::removeRecordStore (const string & *name,* const string & *parentDir*) throw (Error::ObjectDoesNotExist, Error::StrategyError) `[static]`

Remove a RecordStore by deleting all persistant data associated with the store.

Parameters

`in`	*name*	The name of the existing RecordStore.
`in`	*parentDir*	Where, in the file system, the store is rooted.

Exceptions

Er-ror::ObjectDoesNotE	A record with the given key does not exist.
Er-ror::StrategyError	An error occurred when using the underlying storage system.

E.24.3.16 static void BiometricEvaluation::IO::RecordStore::mergeRecordStores (const string & *mergedName,* const string & *mergedDescription,* const string & *parentDir,* const string & *type,* RecordStore ∗ *recordStores[ ],* size_t *numRecordStores*) throw (Error::ObjectExists, Error::StrategyError) `[static]`

Create a new RecordStore that contains the contents of several RecordStores.

Parameters

`in`	*merged-Name*	The name of the new RecordStore that will be created.
`in`	*mergedDe-scription*	The text used to describe the RecordStore.
`in`	*parentDir*	Where, in the file system, the new store should be rooted.
`in`	*type*	The type of RecordStore that mergedName should be.
`in`	*record-Stores*	An array of RecordStore∗ that should be merged into mergedName.

in	*num-Record-Stores*	The number of RecordStore∗ in recordStores.

Exceptions

Error::ObjectExists	A RecordStore with mergedNamed in parentDir already exists.
Error::StrategyError	An error occurred when using the underlying storage system.

E.24.3.17 static void BiometricEvaluation::IO::RecordStore::mergeRecordStores (const string & *mergedName*, const string & *mergedDescription*, const string & *parentDir*, const string & *type*, tr1::shared_ptr< RecordStore > *recordStores[ ]*, size_t *numRecordStores*) throw (Error::ObjectExists, Error::StrategyError) `[static]`

Create a new RecordStore that contains the contents of several RecordStores.

Parameters

in	*merged-Name*	The name of the new RecordStore that will be created.
in	*mergedDe-scription*	The text used to describe the RecordStore.
in	*parentDir*	Where, in the file system, the new store should be rooted.
in	*type*	The type of RecordStore that mergedName should be.
in	*record-Stores*	An array of RecordStore shared pointers, such as those returned from IO::Factory, that should be merged into mergedName.
in	*num-Record-Stores*	The number of RecordStore∗ in recordStores.

Exceptions

Error::ObjectExists	A RecordStore with mergedNamed in parentDir already exists.
Error::StrategyError	An error occurred when using the underlying storage system.

E.24.4 Member Data Documentation

E.24.4.1 const string BiometricEvaluation::IO::RecordStore::CONTROLFILENAME `[static]`

The name of the control file, a properties list.

E.24.4.2 const string BiometricEvaluation::IO::RecordStore::NAMEPROPERTY `[static]`

Keys used in the Properties list for the RecordStore.

"Name" - The name of the store "Description" - The description of the store "Count" - The number of items in the store "Type" - The type of RecordStore.

E.24.4.3 const string BiometricEvaluation::IO::RecordStore::BERKELEYDBTYPE `[static]`

The known RecordStore type strings: "BerkeleyDB" - Berkeley database "Archive" - Archive file "File" - One file per record

E.24.4.4 const int BiometricEvaluation::IO::RecordStore::BE_RECSTORE_SEQ_START = 1 `[static]`

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

Parameters

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

Returns

The length of the record currently in sequence.

Exceptions

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

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

- be_io_recordstore.h

E.25 BiometricEvaluation::Error::SignalManager Class Reference

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

```
#include <be_error_signal_manager.h>
```

Public Member Functions

- SignalManager () throw (Error::StrategyError)
- SignalManager (const sigset_t signalSet) throw (Error::ParameterError)
- void setSignalSet (const sigset_t signalSet) throw (Error::ParameterError)
- void clearSignalSet ()
- void setDefaultSignalSet ()
- bool sigHandled ()
- void start () throw (Error::StrategyError)
- void stop () throw (Error::StrategyError)
- void setSigHandled ()
- void clearSigHandled ()

Static Public Attributes

- static bool _canSigJump
- static sigjmp_buf _sigJumpBuf

E.25.1 Detailed Description

A SignalManager object is used to handle signals that come from the operating system. Applications typically do not invoke most methods of a SignalManager, except the setSignalSet(), setDefaultSignalSet(), and sigHandled(). An application wishing to just catch memory errors can simply construct a SignalManager object, and invoke sigHandled() at the end of the signal block to detect whether a signal was handled.

The BEGIN_SIGNAL_BLOCK macro sets up the jump block and tells the SignalManager object to start handling signals. Applications can call either setSignalSet() or setDefaultSignalSet() before invoking these macros to indicate which signals are to be handled.

The END_SIGNAL_BLOCK() macro clears the signal set, so from that point forward application code signals will be handled in the system's default manner until another signal block is created.

A SignalManager is passive (i.e. no signal handlers are installed) until that start() method is called, and becomes passive when stop() is invoked. The signals that are to be handled by the object are maitained as state, and the set of signals can be changed at any time, but are not in effect until start() is called.

Attention

The start(), stop(), setSigHandled() and clearSigHandled() methods are not meant to be used directly by applications, which should use the BEGIN_-SIGNAL_BLOCK()/END_SIGNAL_BLOCK() macro pair.

E.25.2 Constructor & Destructor Documentation

E.25.2.1 BiometricEvaluation::Error::SignalManager::SignalManager ( ) throw (Error::StrategyError)

Construct a new SignalManager object with the default signal handling: SIGSEGV and SIGBUS.

Returns

The SignalManager.

Exceptions

Error::StrategyError	Could not register the signal handler.

E.25.2.2 BiometricEvaluation::Error::SignalManager::SignalManager (const sigset_t *signalSet*) throw (Error::ParameterError)

Construct a new SignalManager object with the specified signal handling, no defaults.

Parameters

signalSet	(in) The signal set; see sigaction(2), sigemptyset(3) and sigaddset(3).

Returns

The SignalManager.

Exceptions

Error::ParameterError	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

E.25.3 Member Function Documentation

E.25.3.1 void BiometricEvaluation::Error::SignalManager::setSignalSet (const sigset_t *signalSet*) throw (Error::ParameterError)

Set the signals this object will manage.

Parameters

signalSet	(in) The signal set; see sigaction(2), sigemptyset(3) and sigaddset(3).

Exceptions

Error::ParameterError	One of the signals in signalSet cannot be handled (SIGKILL, SIGSTOP.).

E.25.3.2 void BiometricEvaluation::Error::SignalManager::clearSignalSet ( )

Clear all signal handling.

E.25.3.3 void BiometricEvaluation::Error::SignalManager::setDefaultSignalSet ( )

Set the default signals this object will manage: SIGSEGV and SIGBUS.

E.25.3.4 bool BiometricEvaluation::Error::SignalManager::sigHandled ( )

Indicate whether a signal was handled.

Returns

true if a signal was handled, false otherwise.

E.25.3.5 void BiometricEvaluation::Error::SignalManager::start ( ) throw (Error::StrategyError)

Start handling signals of the current signal set.

Exceptions

Error::StrategyError	Could not register the signal handler.

Note

If an application invokes start() without setting up a signal jump block, behavior is undefined, and can result in an infinite loop if further processing causes a signal to be raised.

E.25.3.6 void BiometricEvaluation::Error::SignalManager::stop ( ) throw (Error::StrategyError)

Stop handling signals of the current signal set.

Exceptions

Error::StrategyError	Could not register the signal handler.

E.25.3.7 void BiometricEvaluation::Error::SignalManager::setSigHandled ( )

Set a flag to indicate a signal was handled.

E.25.3.8 void BiometricEvaluation::Error::SignalManager::clearSigHandled ( )

Clear the indication that a signal was handled.

E.25.4 Member Data Documentation

E.25.4.1 bool BiometricEvaluation::Error::SignalManager::_canSigJump `[static]`

Flag indicating can jump after handling a signal.

Note

Should not be directly used by applications.

E.25.4.2 sigjmp_buf BiometricEvaluation::Error::SignalManager::_sigJumpBuf `[static]`

The jump buffer used by the signal handler.

Note

Should not be directly used by applications.

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

- be_error_signal_manager.h

E.26 BiometricEvaluation::Process::Statistics Class Reference

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

`#include <be_process_statistics.h>`

Public Member Functions

- Statistics ()
- Statistics (IO::LogCabinet ∗const logCabinet) throw (Error::NotImplemented, Error::ObjectExists, Error::StrategyError)
- void getCPUTimes (uint64_t ∗usertime, uint64_t ∗systemtime) throw (Error::StrategyError, Error::NotImplemented)
- void getMemorySizes (uint64_t ∗vmrss, uint64_t ∗vmsize, uint64_t ∗vmpeak, uint64_t ∗vmdata, uint64_t ∗vmstack) throw (Error::StrategyError, Error::NotImplemented)
- uint32_t getNumThreads () throw (Error::StrategyError, Error::NotImplemented)
- void logStats () throw (Error::ObjectDoesNotExist, Error::StrategyError, Error::NotImplemented)

Create a snapshot of the current process statistics in the LogSheet created in the LogCabinet.

- void startAutoLogging (uint64_t interval) throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError, Error::NotImplemented)

  Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.

- void stopAutoLogging () throw (Error::ObjectDoesNotExist, Error::StrategyError)

  Stop the automatic logging of process statistics.

- void callStatistics_logStats ()

E.26.1 Detailed Description

The Statistics class provides an interface for gathering process statistics, such as memory usage, system time, etc. The information gathered by objects of this class are for the current process, and can optionally be logged to a LogSheet object contained within the provided LogCabinet.

Note

The resolution of a returned value for many methods may not match the resolution allowed by the interface. For example, the operating system my allow for second resolution whereas the interface allows microsecond resolution.

E.26.2 Constructor & Destructor Documentation

E.26.2.1 BiometricEvaluation::Process::Statistics::Statistics ( )

Constructor with no parameters.

E.26.2.2 BiometricEvaluation::Process::Statistics::Statistics (IO::LogCabinet *const *logCabinet*) throw (Error::NotImplemented, Error::ObjectExists, Error::StrategyError)

Construct a Statistics object with the associated LogCabinet.

Parameters

in	*logCabinet*	The LogCabinet obejct where this object will create a LogSheet to contain the statistic information for the process.

Exceptions

Error::NotImplemented	Logging is not supported on this OS. This exception can be thrown when any portion of the statistics gathering cannot be completed.
Error::ObjectExists	The LogSheet already exists. This exception should rarely, if ever, occur.
Error::StrategyError	Failure to create the LogSheet in the cabinet.

E.26.3 Member Function Documentation

E.26.3.1 void BiometricEvaluation::Process::Statistics::getCPUTimes (uint64_t * *usertime,* uint64_t * *systemtime*) throw (Error::StrategyError, Error::NotImplemented)

Obtain the total user and system times for the process, in microseconds. Any of the out parameters can be NULL, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

out	*usertime*	Pointer where to store the total user time.
out	*systemtime*	Pointer where to store the total system time.

Exceptions

Error::StrategyError	An error occurred when obtaining the process statistics from the operating system. The exception information string contains the error reason.
Error::NotImplemented	This method is not implemented on this OS.

E.26.3.2 void BiometricEvaluation::Process::Statistics::getMemorySizes (uint64_t ∗ *vmrss,* uint64_t ∗ *vmsize,* uint64_t ∗ *vmpeak,* uint64_t ∗ *vmdata,* uint64_t ∗ *vmstack*) throw (Error::StrategyError, Error::NotImplemented)

Obtain the current memory set sizes for the process, in kilobytes. Any of the out parameters can be NULL, indicating non-interest in that statistic.

Note

This method may not be implemented in all operating systems.

Parameters

`out`	*vmrss*	Pointer where to store the current resident set size.
`out`	*vmsize*	Pointer where to store the current total virtual memory size.
`out`	*vmpeak*	Pointer where to store the peak total virtual memory size.
`out`	*vmdata*	Pointer where to store the current virtual memory data segment size.
`out`	*vmstack*	Pointer where to store the current virtual memory stack segment size.

Exceptions

Error::StrategyError	An error occurred when obtaining the process statistics from the operating system. The exception information string contains the error reason.
Error::NotImplementec	This method is not implemented on this OS.

E.26.3.3 uint32_t BiometricEvaluation::Process::Statistics::getNumThreads ( ) throw (Error::StrategyError, Error::NotImplemented)

Obtain the number of threads composing this process.

Note

This method may not be implemented in all operating systems.

Exceptions

Error::StrategyError	An error occurred when obtaining the process info from the operating system. The exception information string contains the error reason.
Error::NotImplementec	This method is not implemented on this OS.

E.26.3.4 void BiometricEvaluation::Process::Statistics::logStats ( ) throw (Error::ObjectDoesNotExist, Error::StrategyError, Error::NotImplemented)

Create a snapshot of the current process statistics in the LogSheet created in the LogCabinet.

Exceptions

Error::ObjectDoesNotE	The LogSheet does not exist; this object was not created with LogCabinet object.
Error::StrategyError	An error occurred when writing to the LogSheet.
Error::NotImplementec	The statistics gathering is not implemented for this operating system.

E.26.3.5 void BiometricEvaluation::Process::Statistics::startAutoLogging (uint64_t *interval*) throw (Error::ObjectDoesNotExist, Error::ObjectExists, Error::StrategyError, Error::NotImplemented)

Start logging process statistics automatically, in intervals of microseconds. The first log entry will occur soon after the call to this method as the delay interval is invoked after the first entry.

Note

It is unrealistic to expect that log entries can be made at a rate of one per microsecond.
If stopAutoLogging() is called very soon after the start, a log entry may not be made.

Parameters

`in`	*interval*	The gap between logging snapshots, in microseconds.

Exceptions

Error::ObjectDoesNotE	The LogSheet does not exist; this object was not created with LogCabinet object.
Error::ObjectExists	Autologging is currently invoked.
Error::StrategyError	An error occurred when writing to the LogSheet.
Error::NotImplementec	The statistics gathering is not implemented for this operating system.

E.26.3.6 void BiometricEvaluation::Process::Statistics::stopAutoLogging ( ) throw (Error::ObjectDoesNotExist, Error::StrategyError)

Stop the automatic logging of process statistics.

Exceptions

Error::ObjectDoesNotE	Not currently autologging.
Error::StrategyError	An error occurred when stopping, most likely because the logging thread died.

E.26.3.7 void BiometricEvaluation::Process::Statistics::callStatistics_logStats ( )

Helper function in C++ space that has access to this object, and is called from C space by the logging thread. Applications should not call this function.

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

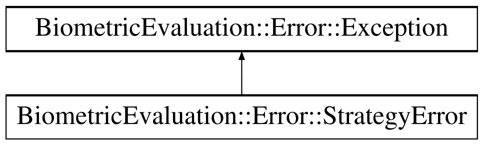
- be_process_statistics.h

E.27 BiometricEvaluation::Error::StrategyError Class Reference

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

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

Inheritance diagram for BiometricEvaluation::Error::StrategyError:


BiometricEvaluation::Error::Exception
BiometricEvaluation::Error::StrategyError

Public Member Functions

- StrategyError ()
- StrategyError (string info)

E.27.1 Detailed Description

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

E.27.2 Constructor & Destructor Documentation

E.27.2.1 BiometricEvaluation::Error::StrategyError::StrategyError ( )

Construct a StrategyError object with the default information string.

Returns

The StrategyError object.

E.27.2.2 BiometricEvaluation::Error::StrategyError::StrategyError (string *info*)

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

Returns

The StrategyError object.

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

- be_error_exception.h

E.28 BiometricEvaluation::Time::Timer Class Reference

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

```
#include <be_time_timer.h>
```

Public Member Functions

- Timer ()
- void start () throw (Error::StrategyError)
- void stop () throw (Error::StrategyError)
- uint64_t elapsed () throw (Error::StrategyError)

E.28.1 Detailed Description

This class can be used by applications to report the amount of time a block of code takes to execute. Applcations wrap the block of code in the Timer::start() and Timer::stop() calls, then use Timer::elapsed() to obtain the calculated time of the operation.

E.28.2 Constructor & Destructor Documentation

E.28.2.1 BiometricEvaluation::Time::Timer::Timer ( )

Constructor for the Timer object.

E.28.3 Member Function Documentation

E.28.3.1 void BiometricEvaluation::Time::Timer::start ( ) throw (Error::StrategyError)

Start tracking time.

Exceptions

Error::StrategyError	This object is currently timing an operation or an error occurred when obtaining timing information.

E.28.3.2 void BiometricEvaluation::Time::Timer::stop ( ) throw (Error::StrategyError)

Stop tracking time.

Exceptions

Error::StrategyError	This object is not currently timing an operation or an error occurred when obtaining timing information.

E.28.3.3 uint64_t BiometricEvaluation::Time::Timer::elapsed ( ) throw (Error::StrategyError)

Get the elapsed time in microseconds between calls to this object's start() and stop() methods.

Returns

The number of microseconds between calls to this object's start() and stop() methods.

Exceptions

Error::StrategyError	This object is currently timing an operation or an error occurred when obtaining timing information.

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

- be_time_timer.h

E.29 BiometricEvaluation::Time::Watchdog Class Reference

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

```
#include <be_time_watchdog.h>
```

Public Member Functions

- Watchdog (const uint8_t type) throw (Error::ParameterError)
- void setInterval (uint64_t interval)
- void start () throw (Error::StrategyError)
- void stop () throw (Error::StrategyError)
- bool expired ()
- void setCanSigJump ()
- void clearCanSigJump ()
- void setExpired ()
- void clearExpired ()

Static Public Attributes

- static const uint8_t PROCESSTIME = 0
- static const uint8_t REALTIME = 1
- static bool **_canSigJump**
- static sigjmp_buf **_sigJumpBuf**

E.29.1 Detailed Description

A Watchdog object can be used by applications to limit the amount of processing time taken by a block of code. A Watchdog object is used to set a timer that, upon expiration, will force a jump to a location within the process. An application can detect whether the timer expired at that point in the code. Watchdog builds on the POSIX setitimer(2) call. Timer intervals are in terms of process virtual time or real time, based on how the object is constructed.

Most applications will not directly invoke the methods of the WatchDog class, instead using the BEGIN_WATCHDOG_BLOCK() and END_WATCHDOG_BLOCK() macros. Applications should not install there own signal handlers, but use the SignalManager class instead.

The BEGIN_WATCHDOG_BLOCK macro sets up the jump block and tells the Watchdog object to start handling the alarm signal. Applications must call setInterval() before invoking the BEGIN_WATCHDOG_BLOCK() macro.

The END_WATCHDOG_BLOCK() macro disables the watchdog timer, but doesn't affect the current interval value. Applications can set the interval once and use the BEGIN/END block macros repeatedly. Failure to call setInterval() results in an effectively disabled timer, as does setting the interval to 0.

Note

Process virtual timing may not be available on all systems. In those cases, an application compilation error will occur because PROCESSTIME will not be defined.

Attention

On many systems, the sleep(3) call is implemented using alarm signals, the same technique used by the Watchdog class. Therefore, applications should not call sleep(3) inside the Watchdog block; behavior is undefined in that case, but usually results in cancellation of the Watchdog timer.
The setCanSigJump(), clearCanSigJump(), setExpired() and clearExpired() methods are not meant to be used directly by applications, which should use the BEGIN_WATCHDOG_BLOCK()/END_WATCHDOG_BLOCK() macro pair.

See also

Error::SignalManager

E.29.2 Constructor & Destructor Documentation

E.29.2.1 BiometricEvaluation::Time::Watchdog::Watchdog (const uint8_t *type*) throw (Error::ParameterError)

Construct a new Watchdog object.

Parameters

`in`	*type*	The type of timer, ProcessTime or RealTime.

Returns

The Watchdog object.

Exceptions

Error::ParameterError	The type is invalid.

E.29.3 Member Function Documentation

E.29.3.1 void BiometricEvaluation::Time::Watchdog::setInterval (uint64_t *interval*)

Set the interval for the timer, but don't start the timer. Setting a value of 0 will essentially disable the timer. Timer intervals are in microseconds, however actual intervals are dependent on the resolution of the system clock, and may not be at microsecond resolution.

Parameters

`in`	*interval*	The timer interval, in microseconds.

E.29.3.2 void BiometricEvaluation::Time::Watchdog::start ( ) throw (Error::StrategyError)

Start a watchdog timer.

Exceptions

Error::StrategyError	Could not register the signal handler, or could not create the timer.

E.29.3.3 void BiometricEvaluation::Time::Watchdog::stop ( ) throw (Error::StrategyError)

Stop a watchdog timer.

Exceptions

Error::StrategyError	Could not clear the timer.

E.29.3.4 bool BiometricEvaluation::Time::Watchdog::expired ( )

Indicate whether the watchdog timer expired.

Returns

true if the timer expired, false otherwise.

E.29.3.5 void BiometricEvaluation::Time::Watchdog::setCanSigJump ( )

Indicate that the signal handler can jump into the application code after handling the signal.

E.29.3.6 void BiometricEvaluation::Time::Watchdog::clearCanSigJump ( )

Clears the flag for the Watchdog object to indicate that the signal jump block is no longer valid.

E.29.3.7 void BiometricEvaluation::Time::Watchdog::setExpired ( )

Set a flag to indicate the timer expired.

E.29.3.8 void BiometricEvaluation::Time::Watchdog::clearExpired ( )

Clear the flag indicating the timer expired.

E.29.4 Member Data Documentation

E.29.4.1 const uint8_t BiometricEvaluation::Time::Watchdog::PROCESSTIME = 0 `[static]`

A Watchdog based on process time.

E.29.4.2 const uint8_t BiometricEvaluation::Time::Watchdog::REALTIME = 1 `[static]`

A Watchdog based on real (wall clock) time.

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

- be_time_watchdog.h

Index