NPS-CS-11-007

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

IMPLEMENTATION OF LIBEWFCS

by

Bruce Allen

September 26, 2011

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Daniel T. Oliver
President

Leonard A. Ferrari
Executive Vice President and
Provost

This report was prepared for and funded by the Defense Intelligence Agency, Washington, DC.

This report was prepared by:

Bruce Allen
Research Associate
Department of Computer Science

Reviewed by:

Peter Denning
Chairman
Department of Computer Science

Released by:

Karl A. van Bibber, Ph.D.
Vice President and Dean of Research

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–9–2011	Technical Report	2010-10-01—2011-09-30

4. TITLE AND SUBTITLE

Implementation of libewfcs

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Bruce Allen

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

NPS-CS-11-007

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DIA

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for Public Release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

The views expressed in this report are those of the author and do not necessarily reflect the official policy or position of the Department of Defense or the U.S. Government.

14. ABSTRACT

`libewfcs` is a a C# library that can read the Expert Witness Format (EWF) `.E01` for computer forensic information such as disk images. `libewfcs` provides portability and security that is not available in existing solutions and also provides a means for cross-validation with existing EWF readers. In addition to providing basic EWF reading services, `libewfcs` permits reading from custom EWF input streams, provides image streaming capability, and internally optimizes performance using caching and aligned reading. This document describes the interfaces and internal operation of `libewfcs`.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	22	

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

Abstract

libewfcs is a a C# library that can read the Expert Witness Format (EWF) .E01 for computer forensic information such as disk images. libewfcs provides portability and security that is not available in existing solutions and also provides a means for cross-validation with existing EWF readers. In addition to providing basic EWF reading services, libewfcs permits reading from custom EWF input streams, provides image streaming capability, and internally optimizes performance using caching and aligned reading. This document describes the interfaces and internal operation of libewfcs.

1 Introduction

This document describes the libewfcs EWF .E01 image reader. Topics include:

- Overview of the EWF .E01 segment file format and how libewfcs reads it.
- How libewfcs is distributed.
- Public interfaces provided by libewfcs.
- Implementation of libewfcs.
- Available EWF .E01 image readers.

2 Overview of the EWF .E01 Segment File Format

EWF segment files contain media images encoded in the EWF format. libewfcs is a reader library for reading image bytes from EWF segment files encoded in the .E01 segment file format. Please refer to the EWF Specification [1] for a detailed description of the EWF format.

2.1 EWF .E01 Segment Files

EWF files formatted in the .E01 format have the following properties:

1. They are serial files, starting with suffix .E01, and are called segment files, see Figure 1.

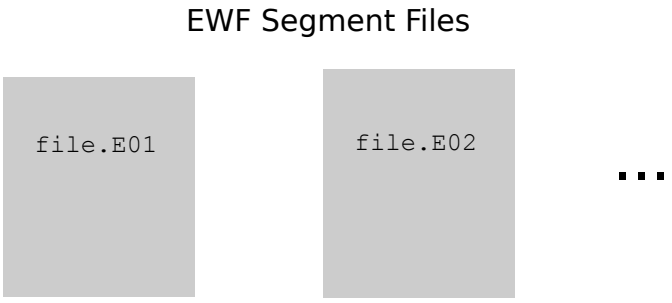

Figure 1: A media image is contained within one or more .E01 segment files.

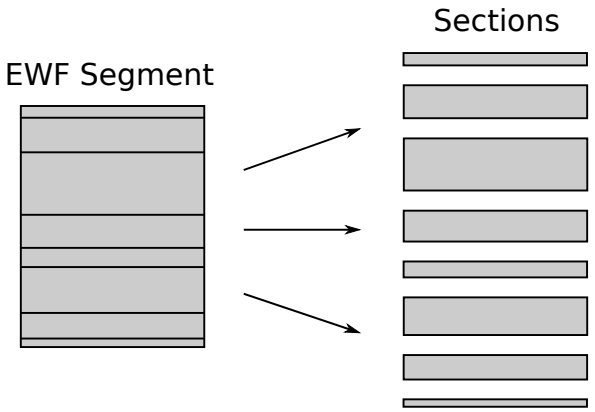

Figure 2: A large segment file contains many sections.

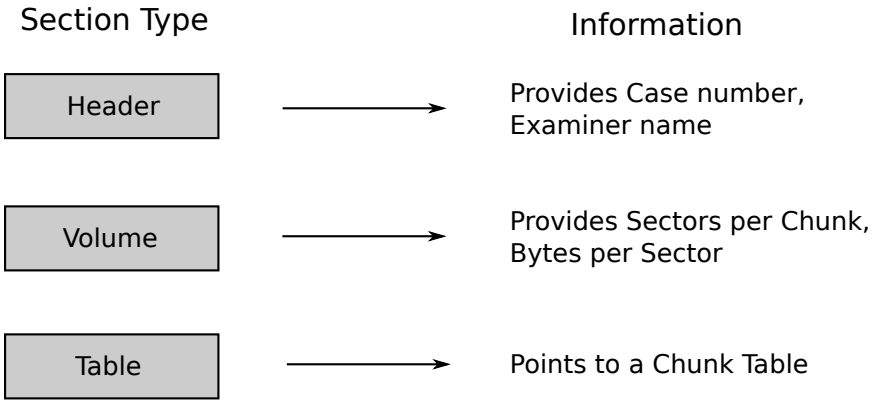

Figure 3: `libewfcs` processes section types to obtain various EWF information.

2. Each segment file contains a magic number (`.E01` file signature) identifying it as a valid EWF file.
3. Data in each segment file is encapsulated in contiguous file divisions called sections. There are many section types. Sections contain the media image and metadata. An example segment file and its sections are depicted in Figure 2.

2.2 Sections

EWF `.E01` segment files are composed of contiguous sections. There are 13 section types. Section types significant to reading image media are shown in Figure 3. They include:

- The Header Section, which contains metadata about the series of EWF segment files such as case number and examiner name.
- The Volume Section, which contains storage information about the segment files, most importantly the sectors per chunk and the number of bytes per sector. These values are used to determine the number of bytes per chunk. This is important because the image is stored in chunks. When reading image data, the chunk size determines which chunk to read from.

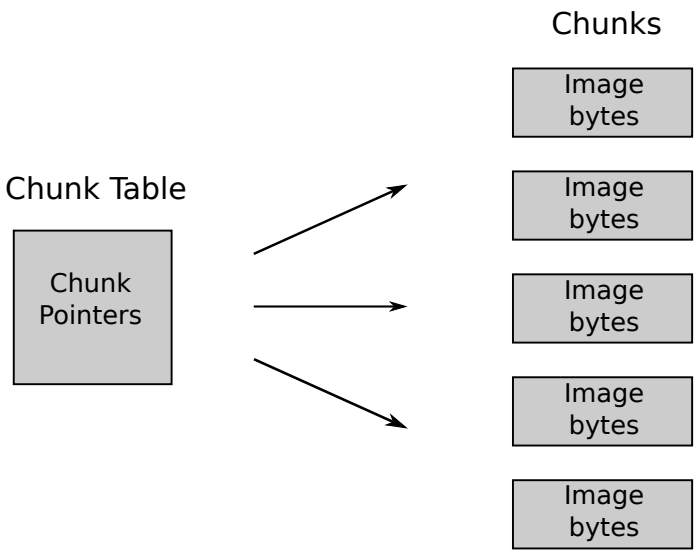

Figure 4: Each Chunk Table points to a number of chunks of image bytes.

- The Table Sections, which define the location and size of Chunk Tables and the chunk offset.

All sections start with the same Section Prefix. The Section Prefix is 72 bytes in size.

2.3 Chunk Tables

Chunk Tables point to chunks which contain image bytes, as shown in Figure 4. There are many Chunk Tables because Chunk Tables and chunks are limited in size. Each entry in a Chunk Table is an integer. The most significant bit (MSB) of the integer is used to indicate whether the associated chunk is compressed, which they typically are. The remaining 31 bits of the integer indicate the start offset in the segment file where the chunk resides. Since segment files can be larger than 2^{31} bytes in size, the chunk offset defined in the Table Section is added to this value to determine the actual start offset in the segment file.

The size of a chunk is determined as follows: If there is another Chunk Table entry, the size is the next entry minus one. If there are no more Chunk Table entries in the Chunk Table, the size is the start of the next section after the chunk start offset minus one.

2.4 Chunks

Chunks contain image bytes. Chunks usually need decompressed, as indicated by the MSB of their Chunk Table entry. All decompressed chunks are the same size except the last chunk, which may be smaller. Since all decompressed chunks are the same size, the chunk index of any image offset is calculated with the equation $ChunkIndex = ImageOffset/ChunkSize$.

3 Distribution

A source code distribution of the libewfcs library is available. The libewfcs library requires external code for inflating compressed zlib data. The requisite portion of this zlib code, obtained from the public domain at Web address http://sourceforge.net/projects/sharpdevelop/files/SharpZipLib/0.86/SharpZipLib_0860_SourceSamples.zip/download, is included with the distribution. The libewfcs Makefile, included, produces a Microsoft Intermediate Language (MSIL) library file of libewfcs in file libewfcs0.dll. The Makefile also compiles the Zlib code required by libewfcs in file libz0.dll. Zlib does not need to be compiled separately.

4 Public Interfaces

4.1 Image Reader Interfaces

libewfcs provides interfaces for creating an EWF image reader instance, reading the image, determining the image size, and reading image properties.

- Constructor EWFImageReader(string filename) creates the EWF image reader for reading EWF files formatted in the .E01 format from the local file system.
- Constructor EWFImageReader(IEWFSegmentStreamProvider ewfSegmentStreamProvider, string segmentName) creates the EWF image reader for reading EWF files formatted in the .E01 format using the custom EWF segment stream provider specified.
- Interface MemoryStream ReadImageBytes (long imageAddress, byte[] buffer, int offset, int count) reads image bytes from the image offset into the buffer provided.
- Interface string GetImageProperties() returns image properties including header information residing in the .E01 file.
- Property ImageSize provides the size, in bytes, of the image file.
- Property ChunkSize provides the size, in bytes, of image chunks.
- Property DisableAdler32Validation may be used to disable Adler32 validation, improving performance at the cost of losing the ability to check data integrity.
- Interface Close() releases the internal file stream resource used by EWFImageReader.
- Constant VersionDate identifies the buld date of libewfcs.

4.2 Image Stream Interfaces

libewfcs provides EWFImageStream for managing the EWF image reader as a stream service.

The following interfaces are supported in addition to the native stream interfaces that this class extends:

- Constructor EWFImageStream (EWFImageReader ewfImageReader) creates an EWF image stream object. It provides its EWF image streaming service by accessing

the EWF image reader specified.
- Please use interface `void OptimizedCopyTo (Stream destinationStream)` to copy the stream rather than using native Stream interface `void CopyTo(Stream destination)`. Although functionally equivalent, the optimized copy is more efficient because it copies EWF chunk-sized data on chunk boundaries. The library would override the native `CopyTo` function, but Stream does not allow it.

4.3 Exception Interfaces

When exceptions occur while reading EWF segment files, an `EWFIOException` is thrown which contains information about the requested EWF segment file read operation that failed.

`EWFIOException` contains the following interfaces:

- Read-only variable `File` identifies the EWF segment file where the attempt to read was made.
- Read-only variable `Address` identifies the location within the EWF segment file where the attempt to read was made.
- Read-only variable `Bytes` contains the bytes that were read, if any. This exception can be thrown when bytes are read but processing is invalid, for example if an Adler32 checksum fails.
- Static interface `string DumpBytes (string message, byte[] bytes, int maxCount)` is provided as a convenience for obtaining a formatted view of up to `maxCount` bytes.

4.4 EWF Segment Stream Provider Interfaces

Although `libewf` typically sources EWF segment file data from EWF segment files contained in the local file system, `libewf` supports the ability to source EWF segment file data from generic streams. To use this capability, instantiate `EWFImageReader` with your own segment stream provider by implementing EWF segment stream provider interface `IEWFSegmentStreamProvider`. Users of this capability must implement the following functions:

- Interface `Stream GetStream (string segmentName)` returns the stream associated with the EWF segment name.
- Interface `bool IsValidFirstSegmentName (string segmentName)` indicates whether the specified segment name is a valid first EWF segment name.
- Interface `string GetNextSegmentName (string previousSegmentName)` returns the next serial segment name in the EWF file naming sequence.
- Interface `void Close ()` closes any resources that this segment stream provider may have open.

5 Implementation of `libewfcs`

5.1 Data

5.1.1 Section Prefix Objects

The library instantiates and caches all Section Prefix objects, one for each Section Prefix defined in the segment files. The library caches all Section Prefix objects, not just Section Prefix objects associated with Volume Sections, because they are required in order to calculate the chunk end pointer when the chunk entry is the last entry in the Chunk Table.

Each Section Prefix object includes the following information:

- The type of section that the section is. The library processes sections differently based on their section type. Example section types are Volume Sections and Table Sections. The library uses only a few of the 13 section types defined.
- The name of the segment file that the Section Prefix is associated with. The library uses this name to identify which segment to open when reading a chunk.
- The chunk index values for the chunks that the section provides, if any. Only Table Sections allocate chunk index values.

5.1.2 Chunks

Chunks are managed as follows: A Table Section defines the start and size of a Chunk Table and also the table base offset used by entries in the Chunk Table. The Chunk Table consists of a contiguous set of integers, where each integer is used to identify the start address of a chunk within the associated segment file. For smaller files, the integer is the address of the chunk in the segment file. For larger files (generated by EnCase v.6+), the chunk address is the integer plus the table base offset defined in the Table Section. The most significant bit of each integer indicates whether the chunk is compressed. Chunks are usually compressed.

5.2 Streams

There are two streams visible in `libewfcs`'s public interfaces:

- EWF Segment Streams provide segment data to the EWF image reader to locate and read information from EWF segments. The `IEWFSegmentStreamProvider` interface allows developers to define their own EWF Segment Streams when EWF segments are not accessible as a file in the file system. `libewfcs` can accept EWF Segment Streams from the file system or from a custom source, as shown in Figure 5.
- The image stream provides media image acces as a Stream.

5.3 Classes

5.3.1 `EWFImageReader`

This class initializes the `EWFImageReader` and provides public reading service interfaces. Most significantly, the `EWFImageReader` provides a public interface for reading an arbitrary amount of image bytes from an arbitrary byte offset. Internally, the `EWFImageReader` must

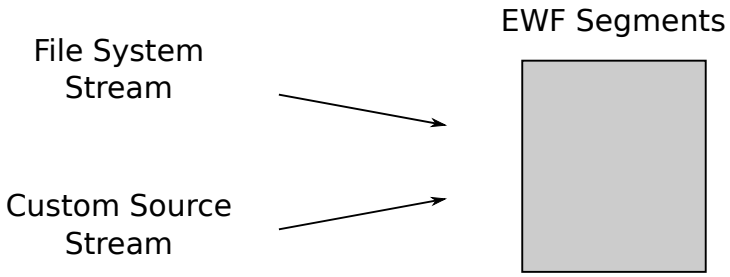

Figure 5: `libewfcs` can obtain EWF segment data from the file system or from a custom source by implementing interface `IEWFSegmentStreamProvider`.

1: verify that the base `.E01` segment file name provided is valid (for segment files the library validates that the file name suffix is `.E01`)
2: **while** Section Prefix entries are available in the segment files **do**
3: read the Section Prefix
4: **if** the section type is a Table Section **then**
5: read the Table Section to determine how many chunk pointers the Chunk Table contains
6: **end if**
7: create a Section Prefix object containing 1) the filename associated with the Section Prefix, 2) the location and size of the Section Prefix, and 3) the number of chunks that the Section Prefix Table points to
8: cache the Section Prefix object in an array for referencing during runtime
9: **end while**
10: determine the chunk size, in bytes, from the Volume Section
11: determine the image size, in bytes, from $(chunkCount - 1) * chunkSize + sizeOfLastChunk$

Figure 6: The initialization process caches all Section Prefix information and determines the chunk size and total media size.

read image bytes from chunks and possibly concatenate bytes from multiple chunks in order to provide the image bytes actually requested.

The `EWFImageReader` initialization process is shown in Figure 6.

The process of reading a chunk of image bytes is shown in Figure 7.

5.3.2 `EWFImageStream`

This class provides `EWFImageReader` as a stream service by subclassing `System.IO.Stream`. Stream method `CopyTo` cannot be overridden, so the library provides the equivalent function as `OptimizedCopyTo (Stream destinationStream)`. This copy function is optimized for EWF by internally reading bytes in chunk sized increments, aligned on chunk size boundaries.

5.3.3 `EWFSegmentFileReader`

This class provides internal accessors for reading content from `.E01` segment files. The following internal interfaces are worth noting:

1: calculate the chunk index: $chunkIndex = imageAddress/chunkSize$
2: find the Section Prefix object containing the chunk index
3: open a file stream for reading the segment file associated with the Section Prefix
4: read the Table Section associated with the Section Prefix object containing the chunk index
5: read the Chunk Table identified by the Table Section
6: calculate the chunk start address within the segment file from the indexed chunk entry with the most significant bit turned off and with the table base offset added, if used
7: **if** this is not the last integer entry in the Chunk Table **then**
8: calculate the chunk end address as the address of the next chunk minus one
9: **else**
10: this is the last integer entry in the Chunk Table, so calculate the chunk end address as the address of the next Section Prefix in the segment file minus one
11: **end if**
12: **if** the chunk is compressed, indicated when the indexed chunk entry integer has its most significant bit set **then**
13: read the chunk using Zlib decompression from the chunk start and end addresses just obtained
14: **else**
15: read the chunk using Adler32 validation from the chunk start and end addresses just obtained
16: **end if**
17: return the image bytes that were just read

Figure 7: The process of reading a chunk of image bytes consists of finding the location of the chunk in its segment file, reading the chunk, and decompressing it.

- `internal byte[] ReadAdler32(string segmentName, long fileOffset, int numBytes)` reads the requested bytes using an Adler32 checksum, checks the checksum, and, if valid, strips out the four checksum bytes and returns the remaining data. This internal interface is used when reading sections.
- `internal byte[] ReadZlib(string segmentName, long fileOffset, int numBytes, int chunkSize)` uses Zlib to read the requested bytes, decompress it, and check its integrity. The chunk size parameter is required so that this interface can pre-allocate output bytes before decompression. This internal interface is used when reading image data that is compressed. When image data is not compressed, as indicated by the most significant bit of the chunk offset not being set, the `ReadAdler32` interface is used instead.
- `internal byte[] ReadRaw(string segmentName, long fileOffset, int numBytes)` provides a raw read with no integrity check. This interface is only used in old `.E01` segment files for reading the Chunk Table, for which the Table Section size is too small to hold the four checksum bytes. When the Table Section size has room to hold the four checksum bytes, Adler32 is used.

The `EWFSegmentFileReader` class also provides accessors for managing `.E01` filenames and for reading basic data types (integers and text) from `.E01` files.

5.3.4 EWFSegmentFileStreamProvider

This class implements interface `IEWFSegmentStreamProvider` in order to provide Stream access to segment files accessed from the local file system. It implements public interface `Stream GetStream(string filename)` for returning a Stream from the specified file in the user's file system. The `.E01` segment file's magic number is validated when the stream is opened. As a performance optimization, this provider keeps the stream open until another stream is requested, at which time the previously open stream is closed and the new stream is opened.

5.3.5 SectionPrefix

This class contains information specific to a given section including the section's associated segment file name and any chunk range covered by the section. During initialization, the associated Section Prefix bytes are read from the segment file to determine the section's type, the start of the next section, and, if the section type is a Table Section, the chunks that this section points to.

The constructor for this class takes the following inputs:

1. `ewfSegmentFileReader`: The segment reader to use for reading the EWF segments.
2. `segmentName`: The name of the EWF segment file associated with this Section Prefix. This name is required for opening the correct section stream.
3. `fileOffset`: The offset into the EWF segment file where the Section Prefix resides.
4. `previousChunkCount`: The number of chunks identified so far. This value is provided to facilitate reading image bytes from chunks. Using it, the library can inspect Section Prefix objects to determine which one contains the chunk number that the library is looking for.

5.3.6 VolumeSection

This class instantiates volume information given a `Stream` and its associated `sectionPrefix`. Most significantly, volume information includes bytes per sector and sectors per chunk values. These values determine the number of image bytes that are in each chunk.

5.3.7 HeaderSection

This class instantiates header information given a `Stream` and its associated `sectionPrefix`. The Header Section contains case information such as the case number and the examiner's name. This class does not facilitate reading image bytes. This class simply provides the ability to read and export Header Section information.

5.3.8 TableSection

This class instantiates Table information given a `Stream` and its associated `sectionPrefix`. The Table Section contains the size of the Chunk Table and the offset of chunk data. The Chunk Table size is used during initialization to index Chunk Table entries. The chunk data offset is used during runtime to calculate the actual chunk data address.

5.3.9 `ChunkTable`

This class reads and instantiates the Chunk Table of a Table Section given a `Stream` and its associated `sectionPrefix`. The Chunk Table is used to obtain the address of the chunk in the `.E01` segment file before reading the actual chunk.

5.3.10 `TableSectionOptimizer`

This optimization caches the last Table Section read so that if the next table section read is the same, the cached Table Section is returned without requesting a physical read.

5.3.11 `ChunkTableOptimizer`

This optimization caches the last Chunk Table read so that if the next Chunk Table read is the same, the cached Chunk Table is returned without requesting a physical read.

5.3.12 `OptimizedInflater`

This class provides an optimization wrapper for the zlib Inflater service. The current optimization implementation is to cache the last compressed and decompressed bytes read, and when the next compressed data is read, if it matches the last compressed value, it returns the decompressed bytes that are cached rather than decompressing again.

`libewfcs` uses Inflater from ICSharpCode.SharpZipLib.Zip.Compression rather than Inflater from Microsoft's DeflateStream in System.IO.Compression because it requires a zlib Inflater, see RFC 1950. The Inflater from Microsoft implements GZip, see RFC 1951, which is incompatible.

5.3.13 `EWFIOException`

This class extends `IOException` by providing properties `File`, `Address`, and `Bytes`, allowing public access to specific properties involved during the exception.

5.4 Interfaces

5.4.1 `IEWFSegmentStreamProvider`

This interface supports implementations of alternate EWF segment file readers. Alternate readers must implement their own segment stream and their own serialized segment naming scheme. For an example implementation of this interface, Please see class `EWFSegmentFileReader` which contains a file system implementation of an EWF segment stream provider.

5.5 Optimizations

The library provides the following optimizations:

1. The last Table Section read is cached so that multiple reads to the same Table Section do not require a physical read.
2. The last Chunk Table read is cached so that multiple reads to the same Chunk Table do not require a physical read.
3. The last compressed and decompressed chunk read is cached so that multiple reads containing the same compressed value do not require running the decompression algorithm.

4. The image stream provider performs its bulk stream copy function using buffered reads of chunk size that are aligned on chunk boundaries.

6 Available EWF .E01 Image Readers

6.1 libewfcs

libewfcs provides the following features:

- Written in C#. As such, it is more portable to .NET platforms and is more secure because it runs as managed code on .NET.
- Able to read EWF .E01 segments from any source, not just the file system. This capability is enabled by implementing EWF segment stream provider interface IEWFSegmentStreamProvider.
- The EWF image may be accessed as a Stream. This capability is available via Stream reader EWFImageReader which wraps and accesses image reader EWFImageReader.
- May be used as a cross-validation tool for comparing with other EWF .E01 image readers.
- Is expected to provide exceptional performance. Performance of libewfcs will be evaluated in FY12.

6.2 libewf

libewf, created by Joachim Metz, is available at http://sourceforge.net/projects/libewf. libewf provides the following features:

- Provides capabilities in addition to reading image bytes from EWF segment files, including the ability to write EWF segment files.
- Written in C.
- Can be wrapped into .NET for extended portability.

6.3 jlibewf

jlibewf, created by Bruce Allen, is also available at http://sourceforge.net/projects/libewf. jlibewf, provides the following features:

- Written in Java. As such, it is more secure because it contains protections provided by running on a JVM.
- Although not measured, jlibewf provides respectable performance.

7 Conclusion

This report shows how libewfcs provides an effective usable open source library for reading media images stored in the EWF .E01 file format. libewfcs is distributed in the form of a libewfcs0.dll .net assembly file. libewfcs provides random access and streaming access to media images. It optimizes performance by caching segment reads and decompressed

bytes. It also offers a custom segment stream reader interface, allowing the ability to read segment data that is available in sources other than the file system.

References

[1] Joachim Metz. Expert witness compression format specification.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Research and Sponsored Programs Office, Code 41
   Naval Postgraduate School
   Monterey, California