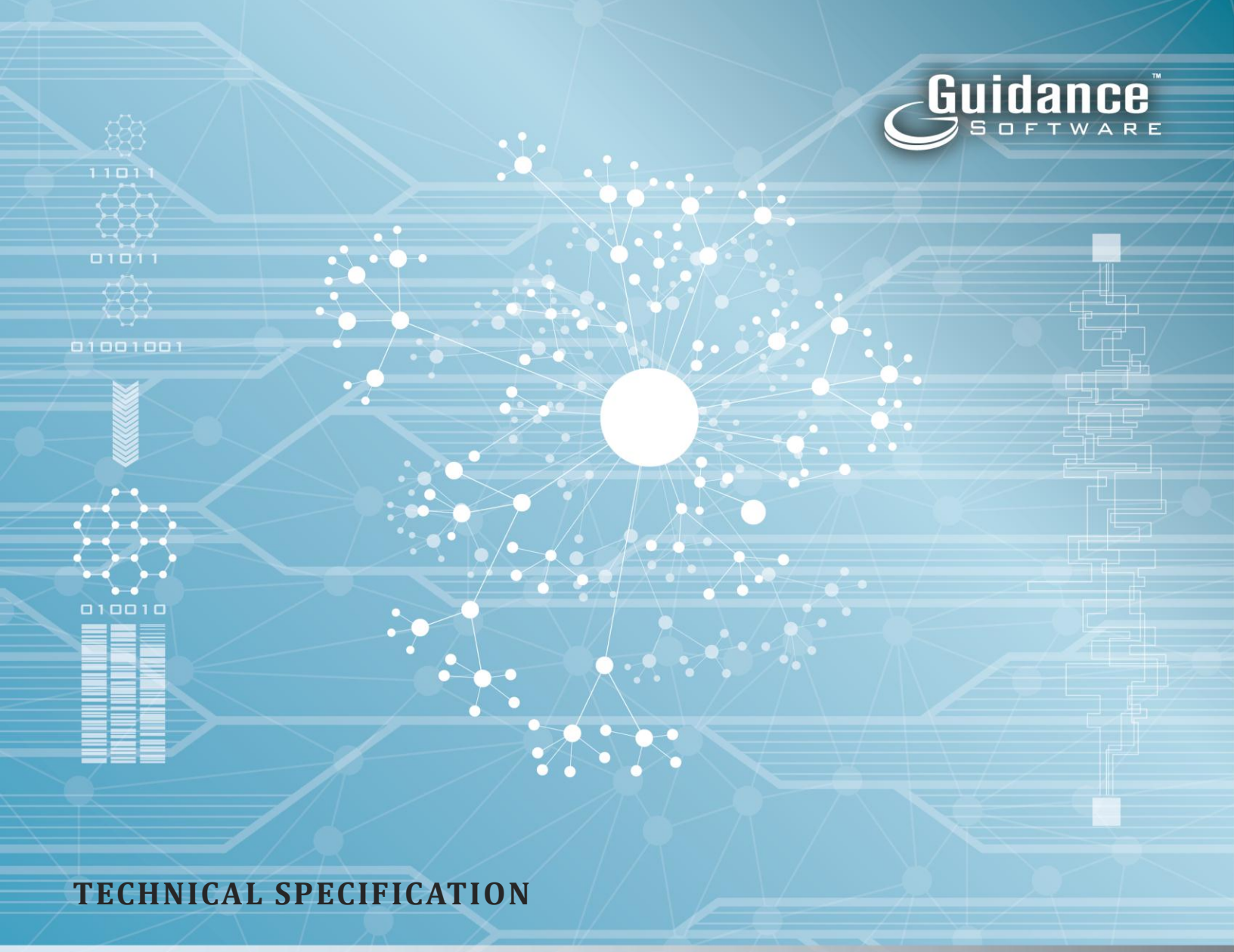


TECHNICAL SPECIFICATION

EnCase Evidence File Format

Version 2

REVISION HISTORY

Revision	Date	Noted Revisions
1.0	12/15/2011	Original Public Release
2.0	08/20/2012	Added additional information and clarified certain areas

Contents

Overview

The existing EnCase evidence file has performed well for over a decade. It is court-validated, well-known, and adopted in the industry. Despite its effectiveness, some limitations remain that can only be overcome with an updated evidence file format.

This document outlines the technical details of the updated EnCase evidence file format version 2 (Ex01) so that developers can customize their applications to integrate with the new format. It describes the details, data structures, and algorithms behind Ex01.

The intended audience of this document is a technical reader with a forensic background and familiarity with C-style binary structure layout and algorithms.

Comparison of E01 and Ex01 Formats

Many of the central design principles of the E01 format have been retained; implementers familiar with the E01 structure will find the Ex01 format similar. The Ex01 format still stores data in blocks that are verified with an individual 32-bit CRC, and all of the source data stored in the file is hashed with the MD5 and/or SHA-1 algorithms if requested by the user. The Ex01 enhancements do not affect features of the file such as these that many courts have relied on to rule that the file is an accepted container of original evidence; the additions merely facilitate the ability to track and handle new characteristics of the stored data.

Ex01 Capabilities

The new Ex01 format introduces the following capabilities:

- Support for encryption of the data.
- Ability to use different compression algorithms.
- Improved support for multi-threaded acquisitions, where sectors can be out of order.
- Efficient storage and handling of sector blocks that are filled with the same pattern (such as 00-byte fills).
- Alignment considerations to improve efficiency and performance.
- Improved support for resuming acquisitions.
- Internal improvements of the data structures.

While some of this new functionality is not yet fully leveraged in the current version, all necessary data is stored, the data structures support expansions, and subsequent versions will use this new format to its fullest.

Evidence File Structure

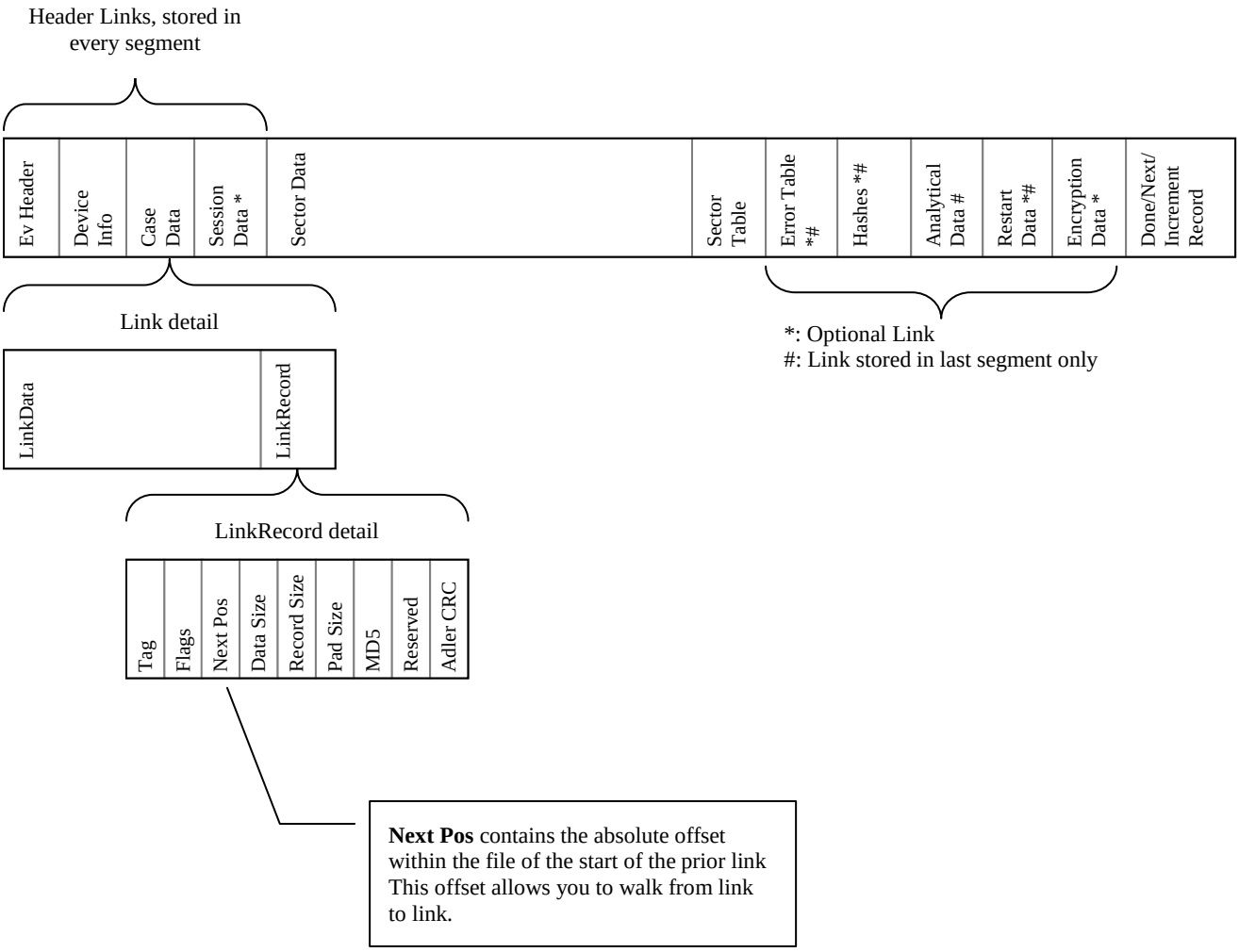
The Ex01 file format is built via segments, which are a type of chunked, binary file. An evidence file may be comprised of one or more segments.

Segments are structured as follows:

- The evidence file **header** (fixed length).
- The header is followed by a series of variable-size structures (**LinkData**), and their associated fixed-size **LinkRecord** structures. LinkRecords are footers that follow the variable-size data. A LinkRecord contains a LinkTag identifying the type of data contained within the Link. Together, the LinkData and LinkRecord are called a **Link**.
- Every segment has a copy of the **Device Information**, **Case Information,** and **Session Table** (optional, if the evidence is a CD/DVD) link. This data is duplicated for recovery purposes.
- These links are followed by the **Sector Data** links.
- At the end of every segment, after the sector data, are the following links:
  - Sector Table.
  - If this *is not* the last segment, then the following links are found:
    - **Encryption Data** (optional, if evidence is encrypted)
    - **Next Record**
  - If this *is* the last segment, then the following links are found:
    - **Error Table** (optional, if errors are encountered)
    - **Hash Records** (optional, if hashes are calculated)
    - **Analytical Data**
    - **Restart Data** (optional, if the evidence file is restarted)
    - **Encryption Data** (optional, if evidence is encrypted)
    - **Done Record**

Although there is a header at the beginning of the link, LinkRecords follow the data they represent so that the file can be written out more efficiently. Each LinkRecord contains the offset of the start of the prior LinkRecord; therefore, EnCase reads evidence files starting from the end of the segment.

Evidence File Segment Layout



Data Structures

Data structures used within the evidence file are all packed with 1-byte packing order. They are padded at the end to multiples of 16 bytes due to encryption considerations. Specific data types used are described in the Data Types section of this document.

Evidence File Header

```
struct EvHeader {
  uint64    Signature;          // 0x00  ASCII string "EVF2\r\n\x81"
  byte      MajorVersion,       // 0x08  EVMAJOR = 2
            MinorVersion;       // 0x09  EVMINOR = 1
  uint16    CompressionEngine;  // 0x0A  The code for the compression engine (CompressionCodes)
  uint32    Series;             // 0x0C  The evidence number in the series (starting at 1)
  GUID      EvGuid;             // 0x10  Random V4 UUID (Unique)
                                // 0x20  Structure size
};
```

Note: EvGuid is the same GUID as in the Device Information Record.

The compression codes currently in use are:

```
enum CompressionCodes {
  COMPRESSION_NONE    = 0,
  COMPRESSION_LZ      = 1,
  COMPRESSION_BZIP2   = 2
};
```

Note: As some links in the evidence file are always compressed, COMPRESSION_NONE will never be used.

Link Record

```
struct LinkRecord {
  uint32   Tag,             // 0x00 Identifies the type of the data stored (enum; detailed in
                            //      the Link Types section of this document)
           Flags;           // 0x04 Flags for the stored data (FlagBits)
  uint64   Next,            // 0x08 The absolute file offset of the next LinkRecord
           DataSize;        // 0x10 Size of the data stored in this record (excluding RecordSize,
                            //      but includingPadSize)
  uint32   RecordSize,      // 0x18 Size of the LinkRecord
           PadSize;         // 0x1C Number of pad bytes in data area to reach the 16-byte
                            //      boundary
  MD5Class MD5Hash;         // 0x20 MD5 hash of the data as stored (if MD5HASHED is set).
                            //      Used to verify the contents of an encrypted evidence
                            //      file without requiring the decryption key
  uint32   ReservedNull[3], // 0x30 Set to 0
           Adler32;         // 0x3C Adler32 value of the previous fields
                            // 0x40 Structure size
};
```

The flags currently in use are:

```
enum FlagBits {
       MD5HASHED   = 1,
       ENCRYPTED   = 2
};
```

Table Structures

The Error Table and the Session Table, stored in links in the evidence file, start with this fixed-size structure prior to the table records:

```
struct TableRecord {
    uint32 Count,           // 0x00 The number of records in this table
           ReservedNull[3]; // 0x04 Unused, set to 0
                            // 0x10 Structure size
};
```

This is followed by the row data for the table. The row size is fixed per type of table. The Adler32 value of the data then follows.

Link Types

Currently available tags are:

- **Device Information: LinkTag (0x01)**

  The Device Information is stored in file object format. A copy of the Device Information is stored in each piece of the evidence file. The end result is stored compressed.

- **Case Data: LinkTag (0x02)**

  Case data is stored in file object format. A copy of the case data is stored in each piece of the evidence file. The link is stored compressed.

- **Sector Data: LinkTag (0x03)**

  Sector data is the core "payload" of the evidence file. Sector data is stored in a series of blocks in this section. Each block must be zero-padded out to 16-byte multiples. Each block is compressed with the compression level stored in the Case Information. If no compression is used on a block, it must have an Adler32 value of the data following the stored data. Compressed data have their own inherent validation.

  Note: These blocks might be out of order.

- **Sector Table: LinkTag (0x04)**

  The Sector Table is a table of BlockOffsets for each sector block in the evidence file. The location in the table is the reference for the block number. This link is not compressed. This link first contains a single BlockTableRecord, followed by an array of BlockOffsets

```
struct BlockTableRecord {
  uint64 TableStart;   // 0x00 First sector block in this table
  uint32 Count,        // 0x08 Number of sector blocks in this table
         ReservedNull;
};

 struct BlockOffset {
    union {
      uint64 FileOffset,  // 0x00 File offset of the start of the block's data.
             FillPattern; // 0x00 uint64 fill pattern if PATTERNFILL is set in Flags
    };
    uint32 Size,          // 0x08 Number of bytes in the file for this block.
           Flags;         // 0x0C Flags for this sector block (FlagBits)
                          // 0x10 Structure size
 };
```

  The flags currently in use are:

```
 enum FlagBits {
   COMPRESSED  = 1, // The data is compressed with the algorithm specified in the file
                    // header
   CHECKSUMED  = 2, //  An Adler32 value of the data is stored at the end
   PATTERNFILL = 4, // FillPattern is the pattern; the data is not stored
 };
```

  Note: PATTERNFILL sectors are inherently compressed, and therefore the COMPRESSED flag should also be set. Uncompressed sectors will never use the PATTERNFILL flag; therefore this flag should be ignored if present without the COMPRESSED flag.

- **Error Table: LinkTag (0x05)**

  The Error Table is a table of sectors where an error reading occurred from the original device. This link is not compressed.

```
struct ErrorRecord {
    uint64 Sector;        // 0x00 The start sector of the error reading
    uint32 Count;         // 0x08 The number of sectors where there was an error reading
    uint32 ReservedNull;  // 0x0C May be used for error code in the future
                          // 0x10 Structure size
};
```

- **Session Table: LinkTag (0x06)**

  The Session Table is a table of records for storing the track information of a CD-ROM. This link is not compressed.

```
struct SessionRecord {
   uint64 Start;           // 0x00 Start sector of track
   uint32 Flags,           // 0x08 Set to 1 if this record describes a music track,
                           //      otherwise 0
          ReservedNull[5]; // 0x0C Set to 0 for expansion
                           // 0x20 Structure size
 };
```

  Note: acquisition of audio tracks is not supported by EnCase. The data in audio tracks is always zeroed out, and the sector size is forced to 2048 bytes.

- **Increment Data: LinkTag (0x07)**

  An Increment Data record is placed near the end of the last evidence file, if the acquisition is not completed. The Increment Data record stores information to allow a restart of the acquisition and show that this is not the last file in the set. The data is stored in file object format. Note that this record is written only if the acquisition is explicitly terminated via cancellation or device dropout and not in the event of a crash during acquisition. The link is stored compressed.

- **MD5 Data: LinkTag (0x08)**

  This is the MD5 hash of the sector data (16 bytes), followed by an Adler32 value of the MD5 data. This link is not compressed.

- **SHA1 Data: LinkTag (0x09)**

  This is the SHA1 hash of the sector data (20 bytes), followed by an Adler32 value of the SHA1 data. This link is not compressed.

- **Restart Data: LinkTag (0x0A)**

  Restart Data is a list of times the acquisition has been restarted. This information is stored in file object format. The link is stored compressed.

- **Encryption Keys Data: LinkTag (0x0B)**

  Encryption Keys Data contains a copy of the keys in each file of the set. Refer to the document outlining the encryption support for Ex01 for further detail.

- **Next Record: LinkTag (0xD)**

  A LinkRecord with the **Next** tag is placed at the end of all but the last evidence file of the set. No data follow this record.

- **Final Info: LinkTag (0xE)**

  This is currently unused.

- **Done: LinkTag (0xF)**

  A LinkRecord with the **Done** tag is placed at the end of the last evidence file of the set. No data follow this record.

- **Analytical Data: LinkTag (0x10)**

  Analytical Data is stored in file object format. A copy of the Analytical Data is stored in each piece of the evidence file. The end result is stored compressed.

File Object Based Storage

Some objects, especially ones that are stored in a tree, are serialized as a **File Object**. File objects serialize data in an object-oriented manner. The data is written out as UTF-16 with a BOM. At the beginning, every field has a short Unicode "tag" to identify its type. This is not repeated for multiple objects of the same type. U+0009 (tab) and U+000D (linefeed) are used to delimit fields and objects. This is the same system that was used in E01.

Escaping Control Characters

Because U+0009 (tab) and U+000A (linefeed) are used to delimit data, these characters must be escaped when they occur in text data. U+000D (carriage return) is also escaped. The following substitutions are used:

- Tab = 0x03
- CR = 0x02
- LF = 0x01

File Object Data

The beginning of the complete File Object contains a UTF16 BOM (0xFF 0xFE). The first item is a number indicating how many file objects are to be written. Each file object has a unique tag to identify it, followed by a linefeed, then the data.

Record File Object

This is a file object that contains a tab-delimited list of records beneath it. The first line of the text is the ordering of the fields by tag (a unique string for each object), followed by a tab-delimited list of data, ending with a linefeed. The end of the records is specified by an empty linefeed.

Text File Object

Contains the UTF16 text of the data.

Integer32

The integer of the data, in text, between 0 and 4,294,967,295 without commas.

Integer64

The integer of the data, in text, between 0 and 18,446,744,073,709,551,615 without commas.

Date

An Integer32 with the number of seconds since January 1, 1970.

IntegerArray

A space-separated list of integers, without commas.

Enum File Object

A single character representing an enumeration.

Bool Property File Object

A single character: "1" if the property is true, blank otherwise.

An example of a file object in binary form is provided in the File Object Detail section for **Case Data: LinkTag (0x02)**

File Object Detail

Device Information: LinkTag (0x01)

The Device Information Link contains information specific to the device being acquired.

DeviceInformation: Record File Object – Tag("**main**")

- Parent object
- Members

Name	File Object Type	Tag	Description
SerialNumber	Text File Object	sn	Serial number of the drive acquired.
Model	Text File Object	md	Model number of the drive acquired.
TotalSectors	Integer64	ts	Total number of sectors of the acquired device.
HPASectors	Integer64	hs	Total number of sectors in the HPA protected section of the drive.
DCOSectors	Integer64	dc	Total number of sectors in the DCO protected section of the drive.
RamSectors	Integer32	rs	Total number of sectors for a Palm RAM device.
LogSector	Integer32	ls	Total number of sectors in the SMART or general logs. (Note: General logs are sectors returned by the READ_LOG_EXT ata command.)
ProcessId	Integer32	pid	The process ID of acquired memory.
DriveType	Enum File Object	dt	The type of the device stored in the evidence file. **r** = Removable **f** = Fixed **c** = CD-ROM **a** = RAM Disk **p** =Palm **l** = Logical evidence **m** = Memory
Label	Text File Object	lb	Description of device.
Physical	Bool Property File Object	ph	True, if the acquired device is physical.
BytesPerSector	Integer32	bp	The number of bytes in each sector of the device.

Case Data: LinkTag (0x02)

The Case Data Link contains information specific to this particular acquisition of the device.

CaseInformation: Record File Object – Tag("**main**")

- Parent object
- Members

Name	File Object Type	Tag	Description
CaseNumber	Text File Object	cn	Case number, as entered by the user.
EvidenceNumber	Text File Object	en	Evidence number, as entered by the user.
Name	Text File Object	nm	Name of evidence, as entered by the user.
Examiner	Text File Object	ex	Name of examiner.
Notes	Text File Object	nt	User-entered notes.
AppVersion	Text File Object	av	Program name and version of acquisition program.
OSVersion	Text File Object	os	Version of operating system that acquisition is performed on.
TargetTime	Date	tt	GMT time of machine that acquisition is performed on.
ActualTime	Date	at	GMT time, as entered by the user.
TotalBlocks	Integer64	tb	Total blocks of evidence file.
Compression	Integer32	cp	Compression used for evidence.
SectorsPerBlock	Integer32	sb	Sectors per block of compressed data.
Granularity	Integer32	gr	Error granularity when reading device.
BlockerFlags	Integer32	wb	Which write blocker is being used during acquisition: 1 = FastBloc 2 = Tableau

This example shows the uncompressed binary format of an example Case Data file object. This data is compressed before being written to the evidence file.

```
ff fe 31 00 0a 00 6d 00 61 00 69 00 6e 00 0a 00 6e 00 6d 00 09 00 63 00 6e 00 09 00 65 00 6e  ÿþ1...m.a.i.n...n.m...c.n...e.n
00 09 00 65 00 78 00 09 00 6e 00 74 00 09 00 61 00 76 00 09 00 6f 00 73 00 09 00 74 00 74 00  ...e.x...n.t...a.v...o.s...t.t.
09 00 61 00 74 00 09 00 74 00 62 00 09 00 63 00 70 00 09 00 73 00 62 00 09 00 67 00 72 00 09  ..a.t...t.b...c.p...s.b...g.r..
00 77 00 62 00 0a 00 46 00 32 00 09 00 09 00 09 00 09 00 09 00 09 00 37 00 2e 00 33 00 2e 00 30 00  .w.b...F.2..........7...3...0.
2e 00 36 00 30 00 09 00 57 00 69 00 6e 00 64 00 6f 00 77 00 73 00 20 00 37 00 09 00 31 00 33  ..6.0...W.i.n.d.o.w.s. .7...1.3
00 32 00 30 00 30 00 37 00 37 00 38 00 35 00 30 00 09 00 31 00 33 00 32 00 30 00 30 00 37 00  .2.0.0.7.7.8.5.0...1.3.2.0.0.7.
37 00 38 00 35 00 30 00 09 00 31 00 31 00 38 00 34 00 30 00 09 00 09 00 36 00 34 00 09 00 36  7.8.5.0...1.1.8.4.0.....6.4...6
00 34 00 09 00 0a 00 0a 00 ì                                                                  .4.......ì
```

The binary data above represents the data in the following table.

main													
nm	cn	en	ex	nt	av	os	tt	at	tb	cp	sb	gr	wb
F2					7.3.0.60	Windows 7	1320077850	1320077850	11840		64	64	

Increment Data: LinkTag (0x07)

Increment Data Link contains information required to restart an acquisition.

IncrementData: Record File Object – Tag("**main**")

- Parent object
- **StateRecord**: Record File Object Data – Tag("**hc**")
  - Parent object - Record for the state of the hash
  - Members

Name	File Object Type	Tag	Description
State0	Integer32	s0	State 0 of the hash.
State1	Integer32	s1	State 1 of the hash.
State2	Integer32	s2	State 2 of the hash.
State3	Integer32	s3	State 3 of the hash.
Count0	Integer32	c0	Count 0 of the hash.
Count1	Integer32	c1	Count 1 of the hash.
Buffer	IntegerArray	buf	The 64-byte buffer for the state of the hash.
Bytes	Integer64	sb	Number of bytes currently hashed.
Key	IntegerArray	sk	Key data for SHA1 Hashing – 64 bytes (array of 16 uint32).
IV	IntegerArray	si	IV data for SHA1 Hashing - 20 bytes (array of 5 uint32).
Options:	Integer32	op	Hashing options (1=MD5, 2=SHA1, 3=Both).

- **IncRecord:** Record File Object Data - Tag("**rec**")
  - Parent object
  - Members

Name	File Object Type	Tag	Description
Start	Integer64	sr	Start sector of this acquisition.
Stop	Integer64	sp	Stop sector of this acquisition.
Date	Date	d	Acquisition start time.

- **ExtraInfo:** Record File Object Data - Tag("**m**") - Record to hold extra information
  - Parent object
  - Members

Name	File Object Type	Tag	Description
Block	Integer64	bl	Block number of written hash state.
TableCount	Integer32	tc	Position in the current sector table of hash state.

Restart Data: LinkTag (0x0A)

This is a list of restart nodes that carry information about when an acquisition was restarted.

– Members

Name	File Object Type	Tag	Description
Props	Integer32	p	Properties of this node
Date	Date	d	Acquisition start time
Start	Integer64	sr	Start sector of this acquisition
Stop	Integer64	sp	Stop sector of this acquisition

The Restart Data are stored as a tree of values, rather than as a single object. File Objects store trees as follows:

- A single line with two integers: <number of immediate children of the top-level parent> 1
- The list of tags used (e.g., "p d sr sp" for Restart Data.
- The nodes of the tree. For each one, two lines are printed:
  - Two integers: 0 <number of immediate children>
  - The values of the various properties

Example:

```
FF FE 31 00 09 00 31 00 0A 00 70 00 09 00 64 00 09 00 73 00 72 00 09 00 73 00 70 00 0A 00 ÿþ1· ·1· ·p· ·d· ·s·r· ·s·p· ·
30 00 09 00 31 00 0A 00 09 00 09 00 09 00 0A 00 30 00 09 00 31 00 0A 00 35 00 09 00 09 00 0· ·1· · · · · ·0· ·1· ·5· · ·
09 00 0A 00 30 00 09 00 30 00 0A 00 09 00 09 00 09 00 31 00 32 00 33 00 34 00 0A 00        · ·0· ·0· · · · ·1·2·3·4· ·
```

1	1		
p	d	sr	sp
0	1		
0	1		
5			
0	0		
			1234

This example consists of a tree of 3 nodes. The parent has 1 child. The child has Props = 5, and no other properties set. It has one (grand)child. The grandchild has no children, and it has Stop set to 1234.

Props corresponds to file properties in the EnCase GUI, and can have the following values:

Name	Value	Meaning
STATEFOLDER	1	Item is a folder (not a leaf).
STATESELECTED	2	The individual item is selected (blue checked).
STATEEXPANDED	4	The individual item is expanded (i.e., its children are shown).
STATEINCLUDE	8	The individual item is included (green plated).

Analytical Data: LinkTag (0x10)

This is a list of data nodes that carry analytical information about the evidence file. This object may be extended in the future.

AnalyticalRecord: Record File Object – Tag("**main**")

- Parent object
- Members

Name	File Object Type	Tag	Description
TotalPatternSpace	Integer64	tps	Total bytes represented by pattern blocks.

Compression

Ex01 supports Lempel Ziv (LZ) and BZIP2 compression format. A given evidence file uses one or the other, but not both; the specific one is specified in the evidence file header.

For LZ compression, the data are stored as RFC1950 format (ZLIB Compressed Data Format Specification). Note that some common compression tools (e.g., .NET's DeflateStream) use RFC1951 format. RFC1950 format consists of a brief header and footer wrapping the compressed RFC1951 data.

For BZIP2 format, the full file format is described on Wikipedia and consists of:

```
.magic:16                       = 'BZ' signature/magic number
.version:8                      = 'h' for Bzip2 ('H'uffman coding), '0' for Bzip1
(deprecated)
.hundred_k_blocksize:8          = '1'..'9' block-size 100 kB-900 kB

.compressed_magic:48            = 0x314159265359 (BCD (pi))
.crc:32                         = checksum for this block
.randomised:1                   = 0=>normal, 1=>randomised (deprecated)
.origPtr:24                     = starting pointer into BWT for after untransform
.huffman_used_map:16            = bitmap, of ranges of 16 bytes, present/not
                                  present
.huffman_used_bitmaps:0..256    = bitmap, of symbols used, present/not present
                                  (multiples of 16)
.huffman_groups:3               = 2..6 number of different Huffman tables in use
.selectors_used:15              = number of times that the Huffman tables are
                                  swapped (each 50 bytes)
*.selector_list:1..6            = zero-terminated bit runs (0..62) of MTF'ed
                                  Huffman table (*selectors_used)
.start_huffman_length:5         = 0..20 starting bit length for Huffman deltas
*.delta_bit_length:1..40        = 0=>next symbol; 1=>alter length
                                                  { 1=>decrement length;
                                                    0=>increment length }
                                                    (*(symbols+2)*groups)
.contents:2..∞                  = Huffman encoded data stream until end of block

.eos_magic:48                   = 0x177245385090 (BCD sqrt(pi))
.crc:32                         = checksum for whole stream
.padding:0..7                   = align to whole byte
```

NOTE: Ex01 files only use the middle part of the BZIP2 format (notated in blue above).

Please note that Ex01 files pad to a full byte, so you can't add the bytes 0x177245385090 and the CRC directly as full bytes. Note that bzip2 does not align the beginning of the footer (.eos_magic and below) on byte boundaries.

For example: an evidence file contains a bzip component with a crc of EF A3 1A AC; the bzip piece in the evidence file ends in 0x80. Running commandline bzip on the same compressed data results in a .bz2 file ending in 8B B9 22 9c 28 48 77 D1 8D 56 00. Only the first bit of the "80" is used in the bzip2; removing that and shifting left by one results in the expected footer 17 72 45 38 50 90 EF A3 1A AC.

Ex01 files should always use the maximum value (9) for the `hundred_k_blocksize` parameter.

Encryption

E01 had the ability of using a "soft" password, meaning that the data itself was not encrypted. Ex01 encrypts the data symmetrically, using AES-256 by default. The encryption key for this can be protected with:

- A password that generates a symmetric key.
- An asymmetric key pair.
- Both of the above.

The encryption algorithms are not fixed. An **algorithmID** is stored so that EnCase can support additional encryption algorithms in the future. Please refer to the document outlining the encryption support for Ex01 for further detail.

Data Types

The data types used are:

Term	Description
byte	Single-byte unsigned 8-bit integer (0-based).
uint16	2-byte unsigned 16-bit integer (0-based) little-endian.
uint32	4-byte unsigned 32-bit integer (0-based) little-endian.
uint64	8-byte unsigned 64-bit integer (0-based) little-endian.
GUID	16-byte UUID V4 based Global Unique Identifier.
MD5Class	16-byte MD5 of data stored in the standard format.
Date	32-bit (4-byte) unsigned value of number of seconds since January 1, 1970 (epoch / Unix time). Specified in UTC.
Text	Unicode data in UTF-16 LE (little-endian) usually 0-terminated. Limited to 3000 characters.
Enum	An integer with specific values.
IntegerArray	An array of 64-bit integers.

Glossary

The **terms** used are:

Term	Description
Link	Binary chunk consisting of the data and a descriptor (LinkRecord).
LinkRecord	Fixed-size footer describing the prior data.
File Object	Text-based storage.
Tag("main")	A UTF16-LE string, in this case "main" (case sensitive).
Segment	One on-disk file; part of a piece of evidence. (For example, Ex01, Ex02, and Ex03 would be three segments of one evidence file.)
E01	Extension for the first segment of evidence in the version 1 format. Sometimes the only segment. Also, a term for a piece of evidence in the evidence file format version 1.
Ex01	Extension for the first segment of evidence in the version 2 format. Sometimes the only segment. Also a term for a piece of evidence in the evidence file format version 2. This was developed for EnCase Version 7 and beyond.
Adler32	A 32-bit checksum used for quick validation purposes.

Support

Guidance Software has published this specification to document a common file format for the storage of digital evidence. Guidance Software does not endorse, support or test any 3rd party implementations of this specifications and makes no warranty as to their viability. Limited development support is provided at the sole discretion of Guidance Software and is available via email at technicalsupport@guidancesoftware.com.