Windows Compound Binary File Format Specification

This document describes the on-disk format of the Compound File, used as the underpinnings of the structured storage support for OLE 2.0.

1. Overview

A Compound File is made up of a number of **virtual streams**. These are collections of data that behave as a linear stream, although their on-disk format may be fragmented. Virtual streams can be user data, or they can be control structures used to maintain the file. Note that the file itself can also be considered a virtual stream.

All allocations of space within a Compound File are done in units called **sectors**. The size of a sector is definable at creation time of a Compound File, but for the purposes of this document will be 512 bytes. A virtual stream is made up of a sequence of sectors.

The Compound File uses several different types of sector: *Fat*, *Directory*, *Minifat*, *DIF*, and *Storage*. A separate type of 'sector' is a *Header*, the primary difference being that a Header is always 512 bytes long (regardless of the sector size of the rest of the file) and is always located at offset zero (0). With the exception of the header, sectors of any type can be placed anywhere within the file. The function of the various sector types is discussed below.

In the discussion below, the term **SECT** is used to describe the location of a sector within a virtual stream (in most cases this virtual stream is the file itself). Internally, a SECT is represented as a ULONG.

2. Sector Types

```
[4 bytes]       typedef unsigned long ULONG;
[2 bytes]       typedef unsigned short USHORT;
[2 bytes]       typedef short OFFSET;
[4 bytes]       typedef ULONG SECT;
[4 bytes]       typedef ULONG FSINDEX;
[2 bytes]       typedef USHORT FSOFFSET;
[4 bytes]       typedef ULONG DFSIGNATURE;
[1 byte]        typedef unsigned char BYTE;
[2 bytes]       typedef unsigned short WORD;
[4 bytes]       typedef unsigned long DWORD;
[2 bytes]       typedef WORD DFPROPTYPE;
[4 bytes]       typedef ULONG SID;
[16 bytes]      typedef CLSID GUID;

[8 bytes]       typedef struct tagFILETIME {
        DWORD dwLowDateTime;
        DWORD dwHighDateTime;
        } FILETIME, TIME_T;


[4 bytes]       const SECT DIFSECT      = 0xFFFFFFFC;
[4 bytes]       const SECT FATSECT      = 0xFFFFFFFD;
[4 bytes]       const SECT ENDOFCHAIN   = 0xFFFFFFFE;
[4 bytes]       const SECT FREESECT     = 0xFFFFFFFF;
```

2.1 Header

```
struct StructuredStorageHeader {        // [offset from start in bytes, length in bytes]
    BYTE            _abSig[8];          // [000H,08] {0xd0, 0xcf, 0x11, 0xe0, 0xa1, 0xb1, 0x1a, 0xe1} for current version,
                                        // was {0x0e, 0x11, 0xfc, 0x0d, 0xd0, 0xcf, 0x11, 0xe0} on old, beta 2 files (late '92)
                                        // which are also supported by the reference implementation
    CLSID           _clid;              // [008H,16] class id (set with WriteClassStg, retrieved with GetClassFile/ReadClassStg)
    USHORT          _uMinorVersion;     // [018H,02] minor version of the format: 33 is written by reference implementation
    USHORT          _uDllVersion;       // [01AH,02] major version of the dll/format: 3 is written by reference implementation
    USHORT          _uByteOrder;        // [01CH,02] 0xFFFE: indicates Intel byte-ordering
    USHORT          _uSectorShift;      // [01EH,02] size of sectors in power-of-two (typically 9, indicating 512-byte sectors)
    USHORT          _uMiniSectorShift;  // [020H,02] size of mini-sectors in power-of-two (typically 6, indicating 64-byte mini-sectors)
    USHORT          _usReserved;        // [022H,02] reserved, must be zero
    ULONG           _ulReserved1;       // [024H,04] reserved, must be zero
    ULONG           _ulReserved2;       // [028H,04] reserved, must be zero
    FSINDEX         _csectFat;          // [02CH,04] number of SECTs in the FAT chain
    SECT            _sectDirStart;      // [030H,04] first SECT in the Directory chain
    DFSIGNATURE     _signature;         // [034H,04] signature used for transactionin: must be zero. The reference implementation
                                        // does not support transactioning
    ULONG           _ulMiniSectorCutoff;    // [038H,04] maximum size for mini-streams: typically 4096 bytes
```

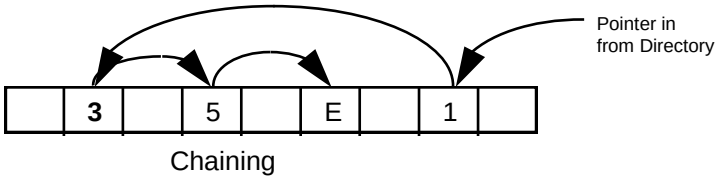
```
SECT        _sectMiniFatStart;  // [03CH,04] first SECT in the mini-FAT chain
FSINDEX     _csectMiniFat;      // [040H,04] number of SECTs in the mini-FAT chain
SECT        _sectDifStart;      // [044H,04] first SECT in the DIF chain
FSINDEX     _csectDif;          // [048H,04] number of SECTs in the DIF chain
SECT        _sectFat[109];      // [04CH,436] the SECTs of the first 109 FAT sectors
};
```

The *Header* contains vital information for the instantiation of a Compound File. Its total length is 512 bytes. There is exactly one *Header* in any Compound File, and it is always located beginning at offset zero in the file.

2.2 Fat Sectors

The **Fat** is the main allocator for space within a Compound File. Every sector in the file is represented within the Fat in some fashion, including those sectors that are unallocated (free). The Fat is a virtual stream made up of one or more Fat Sectors.

Fat sectors are arrays of SECTs that represent the allocation of space within the file. Each stream is represented in the Fat by a **chain**, in much the same fashion as a DOS file-allocation-table (FAT). To elaborate, the set of Fat Sectors can be considered together to be a single array -- each cell in that array contains the SECT of the next sector in the chain, and this SECT can be used as an index into the Fat array to continue along the chain. Special values are reserved for chain terminators (ENDOFCHAIN = 0xFFFFFFFE), free sectors (FREESECT = 0xFFFFFFFF), and sectors that contain storage for Fat Sectors (FATSECT = 0xFFFFFFFD) or DIF Sectors (DIFSECT = 0xFFFFFFC), which are not chained in the same way as the others.



The locations of Fat Sectors are read from the DIF (Double-indirect Fat), which is described below. The Fat is represented in itself, but not by a chain – a special reserved SECT value (FATSECT = 0xFFFFFFFD) is used to mark sectors allocated to the Fat.

A SECT can be converted into a byte offset into the file by using the following formula: SECT << ssheader._uSectorShift + sizeof(ssheader). This implies that sector 0 of the file begins at byte offset 512, not at 0.

2.3 MiniFat Sectors

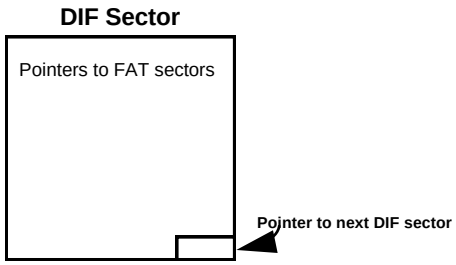
Since space for streams is always allocated in sector-sized blocks, there can be considerable waste when storing objects much smaller than sectors (typically 512 bytes). As a solution to this problem, we introduced the concept of the **MiniFat**. The MiniFat is structurally equivalent to the Fat, but is used in a different way. The virtual sector size for objects represented in the Minifat is 1 << ssheader._uMiniSectorShift (typically 64 bytes) instead of 1 << ssheader._uSectorShift (typically 512 bytes). The storage for these objects comes from a virtual stream within the Multistream (called the **Ministream**).

The locations for MiniFat sectors are stored in a standard chain in the Fat, with the beginning of the chain stored in the header.

A Minifat sector number can be converted into a byte offset into the ministream by using the following formula: SECT << ssheader._uMiniSectorShift. (This formula is different from the formula used to convert a SECT into a byte offset in the file, since no header is stored in the Ministream)

The Ministream is chained within the Fat in exactly the same fashion as any normal stream. It is referenced by the first Directory Entry (SID 0).

2.4 DIF Sectors



The **Double-Indirect Fat** is used to represent storage of the Fat. The DIF is also represented by an array of SECTs, and is chained by the terminating cell in each sector array (see the diagram above). As an optimization, the first 109 Fat Sectors are represented within the header itself, so no DIF sectors will be found in a small (< 7 MB) Compound File.

The DIF represents the Fat in a different manner than the Fat represents a chain. A given index into the DIF will contain the SECT of the Fat Sector found at that offset in the Fat virtual stream. For instance, index 3 in the DIF would contain the SECT for Sector #3 of the Fat.

The storage for DIF Sectors is reserved in the Fat, but is not chained there (space for it is reserved by a special SECT value , DIFSECT=0xFFFFFFFC). The location of the first DIF sector is stored in the header.

A value of ENDOFCHAIN=0xFFFFFFFE is stored in the pointer to the next DIF sector of the last DIF sector.

2.5 Directory Sectors

```
typedef enum tagSTGTY {
    STGTY_INVALID       = 0,
    STGTY_STORAGE       = 1,
    STGTY_STREAM        = 2,
    STGTY_LOCKBYTES     = 3,
    STGTY_PROPERTY      = 4,
    STGTY_ROOT          = 5,
    } STGTY;

typedef enum tagDECOLOR {
    DE_RED              = 0,
    DE_BLACK            = 1,
    } DECOLOR;

struct StructuredStorageDirectoryEntry {        // [offset from start in bytes, length in bytes]
    BYTE        _ab[32*sizeof(WCHAR)];          // [000H,64] 64 bytes. The Element name in Unicode, padded with zeros to
                                                //   fill this byte array
    WORD        _cb;                            // [040H,02] Length of the Element name in characters, not bytes
    BYTE        _mse;                           // [042H,01] Type of object: value taken from the STGTY enumeration
    BYTE        _bflags;                        // [043H,01] Value taken from DECOLOR enumeration.
    SID         _sidLeftSib;                    // [044H,04] SID of the left-sibling of this entry in the directory tree
    SID         _sidRightSib;                   // [048H,04] SID of the right-sibling of this entry in the directory tree
    SID         _sidChild;                      // [04CH,04] SID of the child acting as the root of all the children of this
                                                //   element (if _mse=STGTY_STORAGE)
    GUID        _clsId;                         // [050H,16] CLSID of this storage (if _mse=STGTY_STORAGE)
    DWORD       _dwUserFlags;                   // [060H,04] User flags of this storage (if _mse=STGTY_STORAGE)
    TIME_T      _time[2];                       // [064H,16] Create/Modify time-stamps (if _mse=STGTY_STORAGE)
    SECT        _sectStart;                     // [074H,04] starting SECT of the stream (if _mse=STGTY_STREAM)
    ULONG       _ulSize;                        // [078H,04] size of stream in bytes (if _mse=STGTY_STREAM)
    DFPROPTYPE      _dptPropType;               // [07CH,02] Reserved for future use. Must be zero.
    };
```

The **Directory** is a structure used to contain per-stream information about the streams in a Compound File, as well as to maintain a tree-styled containment structure. It is a virtual stream made up of one or more Directory Sectors. The Directory is represented as a standard chain of sectors within the Fat. The first sector of the Directory chain (the Root Directory Entry)

Each level of the containment hierarchy (i.e. each set of siblings) is represented as a red-black tree. The parent of this set of sibilings will have a pointer to the top of this tree. This red-black tree must maintain the following conditions in order for it to be valid:

1. The root node must always be black. Since the root directory (see below) does not have siblings, it's color is irrelevant and may therefore be either red or black.
2. No two consecutive nodes may both be red.
3. The left child must always be less than the right child. This relationship is defined as:
   - A node with a shorter name is less than a node with a longer name (i.e. compare length of the name)
   - For nodes with the same length names, compare the two names.

The simplest implementation of the above invariants would be to mark every node as black, in which case the tree is simply a binary tree.

A Directory Sector is an array of Directory Entries, a structure represented in the diagram below. Each user stream within a Compound File is represented by a single Directory Entry. The Directory is considered as a large array of Directory Entries. It is useful to note that the Directory Entry for a stream remains at the same index in the Directory array for the life of the stream – thus, this index (called an **SID**) can be used to readily identify a given stream.

The directory entry is then padded out with zeros to make a total size of 128 bytes.

Directory entries are grouped into blocks of four to form Directory Sectors.

2.5.1 Root Directory Entry

The first sector of the Directory chain (also referred to as the first element of the Directory array, or SID 0) is known as the **Root Directory Entry** and is reserved for two purposes: First, it provides a root parent for all objects stationed at the root of the multi-stream. Second, its function is overloaded to store the size and starting sector for the Mini-stream.

The Root Directory Entry behaves as both a stream and a storage. All of the fields in the Directory Entry are valid for the root. The Root Directory Entry's Name field typically contains the string "RootEntry" in Unicode, although some versions of structured storage (particularly the preliminary reference implementation and the Macintosh version) store only the first letter of this string, "R" in the name. This string is always ignored, since the Root Directory Entry is known by its position at SID 0 rather than by its name, and its name is not otherwise used. New implementations should write "RootEntry" properly in the Root Directory Entry for consistency and support manipulating files created with only the "R" name.

2.5.2 Other Directory Entries

Non-root directory entries are marked as either stream (STGTY_STREAM) or storage (STGTY_STORAGE) elements. Storage elements have a _clsid, _time[], and _sidChild values; stream elements may not. Stream elements have valid _sectStart and _ulSize members, whereas these fields are set to zero for storage elements (except as noted above for the Root Directory Entry).

To determine the physical file location of actual stream data from a stream directory entry, it is necessary to determine which FAT (normal or mini) the stream exists within. Streams whose _ulSize member is less than the _ulMiniSectorCutoff value for the file exist in the ministream, and so the _startSect is used as an index into the MiniFat (which starts at _sectMiniFatStart) to track the chain of mini-sectors through the mini-stream (which is, as noted earlier, the standard (non-mini) stream referred to by the Root Directory Entry's _sectStart value). Streams whose _ulSize member is greater than the _ulMiniSectorCutoff value for the file exist as standard streams – their _sectStart value is used as an index into the standard FAT which describes the chain of full sectors containing their data).

2.6 Storage Sectors

Storage sectors are simply collections of arbitrary bytes. They are the building blocks of user streams, and no restrictions are imposed on their contents. Storage sectors are represented as chains in the Fat, and each storage chain (stream) will have a single Directory Entry associated with it.

3. Examples

This section contains a hexadecimal dump of an example structured storage file to clarify the binary file format.

3.1 Sector 0: Header

```
    _abSig                = DOCF 11E0 A1B1 1AE1
    _clid                 = 0000 0000 0000 0000 0000 0000 0000 0000
     uMinorVersion        = 003B
    _uDllVersion          = 3
    _uByteOrder           = FFFE (Intel byte order)
    _uSectorShift         = 9 (512 bytes)
    _uMiniSectorShift     = 6 (64 bytes)
    _usReserved           = 0000
     ulReserved1          = 00000000
    _ulReserved2          = 00000000
    _csectFat             = 00000001
    _sectDirStart         = 00000001
    _signature            = 00000000
    _ulMiniSectorCutoff   = 00001000 (4096 bytes)
     sectMiniFatStart     = 00000002
    _csectMiniFat         = 00000001
     sectDifStart         = FFFFFFFE (no DIF, file is < 7Mb)
    _csectDIF             = 00000000
    _sectFat[]            = 00000000 FFFFFFFF . . . (continues with FFFFFFFF)

000000:  D0CF 11E0 A1B1 1AE1  0000 0000 0000 0000   ................
000010:  0000 0000 0000 0000  3B00 0300 FEFF 0900   ........;.......
000020:  0600 0000 0000 0000  0000 0000 0100 0000   ................
000030:  0100 0000 0000 0000  0010 0000 0200 0000   ................
000040:  0100 0000 FEFF FFFF  0000 0000 0000 0000   ................
000050:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
. . .
0001F0:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
```

3.2 SECT 0: First (Only) FAT Sector

```
    SECT 0: FFFFFFFD = FATSECT: marks this sector as a FAT sector.
              Referred to in header by _sectFat[0]
    SECT 1: FFFFFFFE = ENDOFCHAIN: marks the end of the directory chain,
              referred to in header by _sectDirStart
    SECT 2: FFFFFFFE = ENDOFCHAIN: marks the end of the mini-fat, referred to
              in header by _sectMiniFatStart
    SECT 3: 00000004 = pointer to the next sector in the "Stream 1" data. This sector is
              the first sector of "Stream 1", it is referred to by the Directory Entry
    SECT 4: ENDOFCHAIN (0xFFFFFFFE): marks the end of the "Stream 1" stream data.
    Further Entries are empty (FREESECT = 0xFFFFFFFF)

000200:  FDFF FFFF FEFF FFFF  FEFF FFFF 0400 0000   ................
000210:  FEFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
. . .
0003F0:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
```

3.3 SECT 1: First (Only) Directory Sector

```
    SID 0: Root SID: Root Name = "R"
    SID 1: Element 1 SID: Name = "Storage 1"
    SID 2: Element 2 SID: Name = "Stream 1"
    SID 3: Unused
```

3.3.1 SID 0: Root Directory Entry

```
      _ab          = ("R") (this should be "Root Entry")
      _cb          = 0004 (4 bytes, includes double-null terminator)
      _mse         = 05 (STGTY_ROOT)
      _bflags      = 00 (DE_RED)
      _sidLeftSib  = FFFFFFFF (none)
       sidRightSib = FFFFFFFF (none)
      _sidChild    = 00000001 (SID 1: "Storage 1")
       clsid       = 0067 6156 54C1 CE11 8553 00AA 00A1 F95B
      _dwUserFlags = 00000000 (n/a for STGTY_ROOT)
      _time[0]     = CreateTime   = 0000 0000 0000 0000 (none set)
      _time[1]     = ModifyTime   = 801E 9213 4BB4 BA01 (??)
```

```
      _sectStart   = 00000003 (starting sector of MiniStream)
       ulSize      = 00000240 (length of MiniStream in bytes)
      _dptPropType = 0000 (n/a)

000400:  0052 0000 0000 0000  0000 0000 0000 0000   .R..............
000410:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000420:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000430:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000440:  0400 0500 FFFF FFFF  FFFF FFFF 0100 0000   ................
000450:  0067 6156 54C1 CE11  8553 00AA 00A1 F95B   .gaVT....S.....[
000460:  0000 0000 0000 0000  0000 0000 801E 9213   ................
000470:  4BB4 BA01 0300 0000  4002 0000 0000 0000   K.......@.......
```

3.3.2 SID 1: “Storage 1”

```
      _ab          = ("Storage 1")
       cb          = 0014 (20 bytes, including double-null terminator)
      _mse         = 01 (STGTY_STORAGE)
      _bflags      = 01 (DE_BLACK)
      _sidLeftSib  = FFFFFFFF (none)
       sidRightSib = FFFFFFFF (none)
      _sidChild    = 00000002 (SID 2: “Stream 1”)
       clsid       = 0000 0000 0000 0000 0000 0000 0000 0000 (none set)
      _dwUserFlags = 00000000 (none set)
      _time[0]     = CreateTime   = 00000000 00000000 (none set)
      _time[1]     = ModifyTime   = 00000000 00000000 (none set)
       sectStart   = 00000000 (n/a)
      _ulSize      = 00000000 (n/a)
       dptPropType = 0000 (n/a)

000480:  5300 7400 6F00 7200  6100 6700 6500 2000   S.t.o.r.a.g.e. .
000490:  3100 0000 0000 0000  0000 0000 0000 0000   1...............
0004A0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0004B0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0004C0:  1400 0101 FFFF FFFF  FFFF FFFF 0200 0000   ................
0004D0:  0061 6156 54C1 CE11  8553 00AA 00A1 F95B   .aaVT....S.....[
0004E0:  0000 0000 0088 F912  4BB4 BA01 801E 9213   ........K.......
0004F0:  4BB4 BA01 0000 0000  0000 0000 0000 0000   K...............
```

3.3.3 SID 2: “Stream 1”

```
       ab          = ("Stream 1")
      _cb          = 0012 (18 bytes, including double-null terminator)
      _mse         = 02 (STGTY_STREAM)
      _bflags      = 01 (DE_BLACK)
      _sidLeftSib  = FFFFFFFF (none)
      _sidRightSib = FFFFFFFF (none)
      _sidChild    = FFFFFFFF (n/a for STGTY_STREAM)
      _clsid       = 0000 0000 0000 0000 0000 0000 0000 0000 (n/a)
      _dwUserFlags = 00000000 (n/a)
      _time[0]     = CreateTime   = 00000000 00000000 (n/a)
      _time[1]     = ModifyTime   = 00000000 00000000 (n/a)
       startSect   = 00000000 (SECT in mini-fat, since  ulSize is smaller than  ulMiniSectorCutoff)
      _ulSize      = 00000220 (< ssheader._ulMiniSectorCutoff, so _sectStart is in Mini)
       dptPropType = 0000 (n/a)

000500:  5300 7400 7200 6500  6100 6D00 2000 3100   S.t.r.e.a.m. .1.
000510:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000520:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000530:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000540:  1200 0201 FFFF FFFF  FFFF FFFF FFFF FFFF   ................
000550:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000560:  0000 0000 0000 0000  0000 0000 0000 0000   ................
000570:  0000 0000 0000 0000  2002 0000 0000 0000   ........ .......
000580:  0000 0000 0000 0000  0000 0000 0000 0000   ................
```

3.3.4 SID 3: Unused

```
000590:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0005A0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
```

```
0005B0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0005C0:  0000 0000 FFFF FFFF  FFFF FFFF FFFF FFFF   ................
0005D0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0005E0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
0005F0:  0000 0000 0000 0000  0000 0000 0000 0000   ................
```

3.4 SECT 3: MiniFat Sector

```
    SECT 0: 00000001: pointer to the second sector in the "Stream 1" data. This sector is
              the first sector of "Stream 1", it is referred to by _sectStart of SID 2
    SECT 1: 00000002: pointer to the third sector in the "Stream 1" data. This sector is
              the second sector of "Stream 1", it is referred to in MiniFat SECT 0, above.
     . . .
     SECT 8: FFFFFFFE = ENDOFCHAIN: marks the end of the "Stream 1" data.

    Further Entries are empty (FREESECT = 0xFFFFFFFF)

000600:  0100 0000 0200 0000  0300 0000 0400 0000   ................
000610:  0500 0000 0600 0000  0700 0000 0800 0000   ................
000620:  FEFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
. . .
0007F0:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
```

3.5 SECT 4: MiniStream (Data of "Stream 1")

```
// referred to by SECTs in MiniFat of SECT 3, above

000800:  4461 7461 2066 6F72  2073 7472 6561 6D20   Data for stream
000810:  3144 6174 6120 666F  7220 7374 7265 616D   1Data for stream
000820:  2031 4461 7461 2066  6F72 2073 7472 6561    1Data for strea
. . .
000A00:  7461 2066 6F72 2073  7472 6561 6D20 3144   ta for stream 1D
000A10:  6174 6120 666F 7220  7374 7265 616D 2031   ata for stream 1

// data ends at 000A1F, MiniSector is filled to the end with known data (a copy of the header or
// FFFFFFF to prevent random disk or memory contents from contaminating the file on-disk.

000A20:  0000 0000 0000 0000  3B00 03FF FE00 0900   ........;.......
000A30:  0600 0000 0000 0000  0000 0000 0000 0100   ................
000A40:  D0CF 11E0 A1B1 1AE1  0000 0000 0000 0000   ................
000A50:  0000 0000 0000 0000  003B 0003 FFFE 0009   .........;......
000A60:  0006 0000 0000 0000  0000 0000 0000 0001   ................
000A70:  0000 0001 0000 0000  0000 1000 0000 0002   ................
000A80:  0000 0001 FFFF FFFE  0000 0000 0000 0000   ................
000A90:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
. . .
000BF0:  FFFF FFFF FFFF FFFF  FFFF FFFF FFFF FFFF   ................
```

Index