[MS-CAB]:
Cabinet File Format

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
04/04/2008	0.1.0	Major	Initial Availability.
06/27/2008	1.0.0	Major	Initial Release.
08/06/2008	1.0.1	Editorial	Revised and edited technical content.
09/03/2008	1.0.2	Editorial	Updated references.
12/03/2008	1.0.3	Editorial	Minor editorial fixes.
03/04/2009	1.0.4	Editorial	Revised and edited technical content.
04/10/2009	2.0.0	Major	Updated technical content and applicable product releases.
07/15/2009	3.0.0	Major	Revised and edited for technical content.
11/04/2009	4.0.0	Major	Updated and revised the technical content.
02/10/2010	5.0.0	Major	Updated and revised the technical content.
05/05/2010	5.1.0	Minor	Updated the technical content.
08/04/2010	5.1.0	No change	No changes to the meaning, language, or formatting of the technical content.
11/03/2010	5.1.0	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1 Introduction

This document specifies the **cabinet file** format. Cabinet files are compressed packages containing a number of related files. The format of a cabinet file is optimized for maximum compression. Cabinet files support a number of compression formats, including MSZIP, LZX, or uncompressed. This document does not specify these internal compression formats.

1.1 Glossary

The following terms are defined in [MS-OXGLOS]:

ASCII
cabinet file
cabinet folder
little-endian
Unicode

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[MS-MCI] Microsoft Corporation, "MCI Compression and Decompression", April 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

1.2.2 Informative References

[MS-OXGLOS] Microsoft Corporation, "Exchange Server Protocols Master Glossary", April 2008.

1.3 Overview

Each file stored in a cabinet is stored completely within a single folder. A cabinet file might contain one or more folders, or portions of a folder. A folder can span across multiple cabinets. Such a series of cabinet files form a set. Each cabinet file contains name information for the logically adjacent cabinet files. Each folder contains one or more files.

Throughout this discussion, cabinets are said to contain "files". This is for semantic purposes only. cabinet files actually store streams of bytes, each with a name and some other common attributes. Whether these byte streams are actually files or some other kind of data is application-defined. A cabinet file contains a cabinet header (CFHEADER), followed by one or more **cabinet folder** (CFFOLDER) entries, a series of one or more cabinet file (CFFILE) entries, and the actual

compressed file data in CFDATA entries. The compressed file data in the CFDATA entry is stored in one of several compression formats, as indicated in the corresponding CFFOLDER structure.

The CAB file format has the following limits:

- A maximum uncompressed size of an input file to store in CAB: 0x7FFF8000 bytes.
- A maximum file COUNT: 0xFFFF.
- A maximum size of a created CAB (compressed): 0x7FFFFFFF bytes.
- A maximum CAB-folder COUNT: 0xFFFF.
- A maximum uncompressed data size in a CAB-folder: 0x7FFF8000 bytes.

1.4 Relationship to Protocols and Other Structures

For more information about data compression format, see [MS-MCI].

1.5 Applicability Statement

None.

1.6 Versioning and Localization

None.

1.7 Vendor-Extensible Fields

None.

2 Structures

The types u1, u2, and u4 are used to represent unsigned 8-bit, 16-bit, and 32-bit integer values, respectively. All multi-byte quantities are stored in **little-endian** order, where the least significant byte comes first.

The cabinet file format is specified here using a C-like structure notation, where successive fields appear in the structure sequentially without padding or alignment. Header fields followed by (optional) can be present, depending on the values in the CFHEADER flags byte.

2.1 CFHEADER

The CFHEADER structure provides information about this cabinet file.

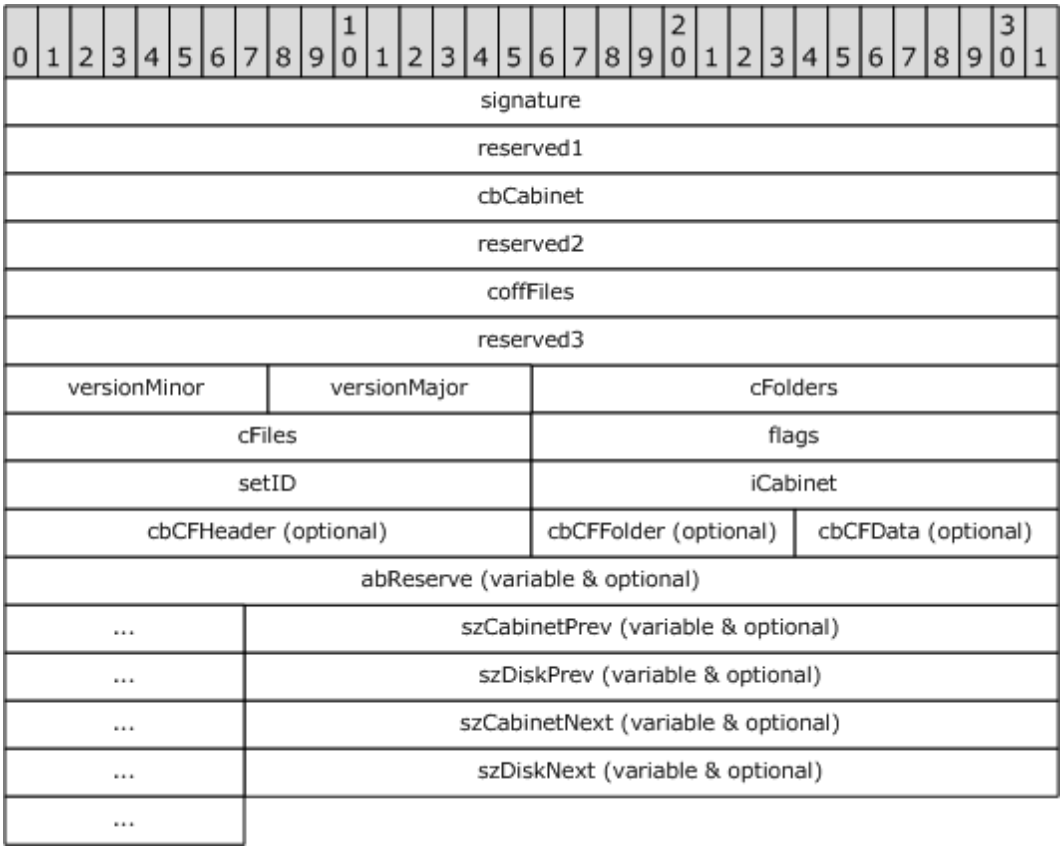

Figure 1: CFFHEADER structure

2.1.1 u1 signature[4]

Contains the characters "M", "S", "C", and "F" (bytes 0x4D, 0x53, 0x43, 0x46). This field is used to ensure that the file is a cabinet file.

2.1.2 u4 reserved1

Reserved field, MUST be set to zero.

2.1.3 u4 cbCabinet

Specifies the total size of the cabinet file, in bytes.

2.1.4 u4 reserved2

Reserved field, MUST be set to zero.

2.1.5 u4 coffFiles

Specifies the absolute file offset, in bytes, of the first CFFILE entry.

2.1.6 u1 versionMinor

Specifies the minor cabinet file format version.

2.1.7 u1 versionMajor

Specifies the major cabinet file format version.

2.1.8 u2 cFolders

Specifies the number of CFFOLDER entries in this cabinet file.

2.1.9 u2 cFiles

Specifies the number of CFFILE entries in this cabinet file.

2.1.10 u2 flags

Specifies bit-mapped values that indicate the presence of optional data:

```
#define cfhdrPREV_CABINET       0x0001
#define cfhdrNEXT_CABINET       0x0002
#define cfhdrRESERVE_PRESENT    0x0004
```

Flags.cfhdrPREV_CABINET is set if this cabinet file is not the first in a set of cabinet files. When this bit is set, the **szCabinetPrev** and **szDiskPrev** fields are present in this CFHEADER.

Flags.cfhdrNEXT_CABINET is set if this cabinet file is not the last in a set of cabinet files. When this bit is set, the **szCabinetNext** and **szDiskNext** fields are present in this CFHEADER.

Flags.cfhdrRESERVE_PRESENT is set if this cabinet file contains any reserved fields. When this bit is set, the **cbCFHeader,cbCFFolder,** and **cbCFData** fields are present in this CFHEADER.

Other bit positions in the flags field are reserved; they SHOULD be set to 0 and MUST be ignored.

2.1.11 u2 setID

Specifies an arbitrarily derived (random) value that binds a collection of linked cabinet files together. All cabinet files in a set will contain the same **setID**. This field is used by cabinet file extractors to ensure that cabinet files are not inadvertently mixed. This value has no meaning in a cabinet file that is not in a set.

2.1.12 u2 iCabinet

Specifies the sequential number of this cabinet in a multi-cabinet set. The first cabinet has iCabinet=0. This field, along with **setID**, is used by cabinet file extractors to ensure that this cabinet is the correct continuation cabinet when spanning cabinet files.

2.1.13 u2 cbCFHeader (optional)

If flags.cfhdrRESERVE_PRESENT is not set, then this field is not present, and the value of **cbCFHeader** MUST be zero. Indicates the size, in bytes, of the **abReserve** field in this CFHEADER. Values for **cbCFHeader** MUST be between 0-60,000.

2.1.14 u1 cbCFFolder (optional)

If flags.cfhdrRESERVE_PRESENT is not set, then this field is not present, and the value of **cbCFFolder** MUST be zero. Indicates the size, in bytes, of the **abReserve** field in each CFFOLDER entry. Values for **cbCFFolder** MUST be between 0-255.

2.1.15 u1 cbCFData (optional)

If flags.cfhdrRESERVE_PRESENT is not set, this field is not present, and the value for **cbCFDATA** MUST be zero. The **cbCFDATA** field indicates the size, in bytes, of the **abReserve** field in each CFDATA entry. Values for **cbCFDATA** MUST be between 0 - 255.

2.1.16 u1 abReserve[cbCFHeader] (optional)

If flags.cfhdrRESERVE_PRESENT is set and **cbCFHeader** is non-zero, then this field contains per-cabinet-file application information. This field is defined by the application, and is used for application-defined purposes.

2.1.17 u1 szCabinetPrev[] (optional)

If flags.cfhdrPREV_CABINET is not set, this field is not present. This is a NULL-terminated **ASCII** string containing the file name of the logically-previous cabinet file. The string can contain up to 255 bytes, plus the NULL byte. Note that this gives the name of the most-recently-preceding cabinet file that contains the initial instance of a file entry. This might not be the immediately previous cabinet file, when the most recent file spans multiple cabinet files. If searching in reverse for a specific file entry, or trying to extract a file that is reported to begin in the "previous cabinet," then **szCabinetPrev** would indicate the name of the cabinet to examine.

2.1.18 u1 szDiskPrev[] (optional)

If flags.cfhdrPREV_CABINET is not set, then this field is not present. This is a NULL-terminated ASCII string containing a descriptive name for the media containing the file named in **szCabinetPrev**, such as the text on the disk label. This string can be used when prompting the user to insert a disk. The string can contain up to 255 bytes, plus the NULL byte.

2.1.19 u1 szCabinetNext[] (optional)

If flags.cfhdrNEXT_CABINET is not set, then this field is not present. This is a NULL-terminated ASCII string containing the file name of the next cabinet file in a set. The string can contain up to 255 bytes, plus the NULL byte. Files extending beyond the end of the current cabinet file are continued in the named cabinet file.

2.1.20 u1 szDiskNext[] (optional)

If flags.cfhdrNEXT_CABINET is not set, then this field is not present. This is a NULL-terminated ASCII string containing a descriptive name for the media containing the file named in **szCabinetNext**, such as the text on the disk label. The string can contain up to 255 bytes, plus the NULL byte. This string can be used when prompting the user to insert a disk.

2.2 CFFOLDER

Each CFFOLDER structure contains information about one of the folders or partial folders stored in this cabinet file. The first CFFOLDER entry immediately follows the CFHEADER entry. CFHEADER.cFolders indicates how many CFFOLDER entries are present.

Folders can start in one cabinet, and continue on to one or more succeeding cabinets. When the cabinet file creator detects that a folder has been continued into another cabinet, it will complete that folder as soon as the current file has been completely compressed. Any additional files will be placed in the next folder. Generally, this means that a folder would span at most two cabinets, but it could span more than two cabinets if the file is large enough.

CFFOLDER entries actually refer to folder fragments, not necessarily complete folders. A CFFOLDER structure is the beginning of a folder if the iFolder value in the first file referencing the folder does not indicate that the folder is continued from the previous cabinet file.

The **typeCompress** field can vary from one folder to the next, unless the folder is continued from a previous cabinet file.

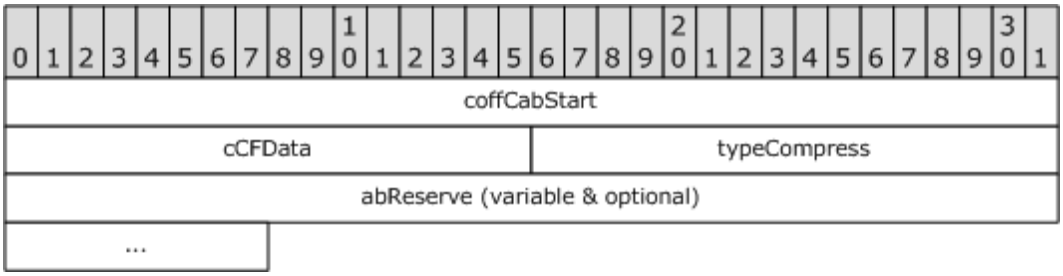

Figure 2: CFFOLDER structure

2.2.1 u4 coffCabStart

Specifies the absolute file offset of the first CFDATA block for the folder.

2.2.2 u2 cCFData

Specifies the number of CFDATA structures for this folder that are actually in this cabinet. A folder can continue into another cabinet and have more CFDATA blocks in that cabinet file. A folder can start in a previous cabinet. This number represents only the CFDATA structures for this folder that are at least partially recorded in this cabinet.

2.2.3 u2 typeCompress

Indicates the compression method used for all CFDATA entries in this folder. The following are the valid values.

```
tcompMASK_TYPE          0x000F     // Mask for compression type
```

```
tcompTYPE_NONE          0x0000     // No compression
tcompTYPE_MSZIP         0x0001     // MSZIP
tcompTYPE_QUANTUM       0x0002     // Quantum
tcompTYPE_LZX           0x0003     // LZX
```

2.2.4 u1 abReserve[CFHEADER.cbCFFolder] (optional)

If CFHEADER. flags.cfhdrRESERVE_PRESENT is set and cbCFFolder is non-zero, then this field contains per-folder application information. This field is defined by the application, and is used for application-defined purposes.

2.3 CFFILE

Each CFFILE entry contains information about one of the files stored (or at least partially stored) in this cabinet. The first CFFILE entry in each cabinet is found at absolute offset CFHEADER.coffFiles. CFHEADER.cFiles indicates how many of these entries are in the cabinet. The CFFILE entries in a cabinet are ordered by iFolder value, and then by uoffFolderStart. Entries for files continued from the previous cabinet will be first, and entries for files continued to the next cabinet will be last.

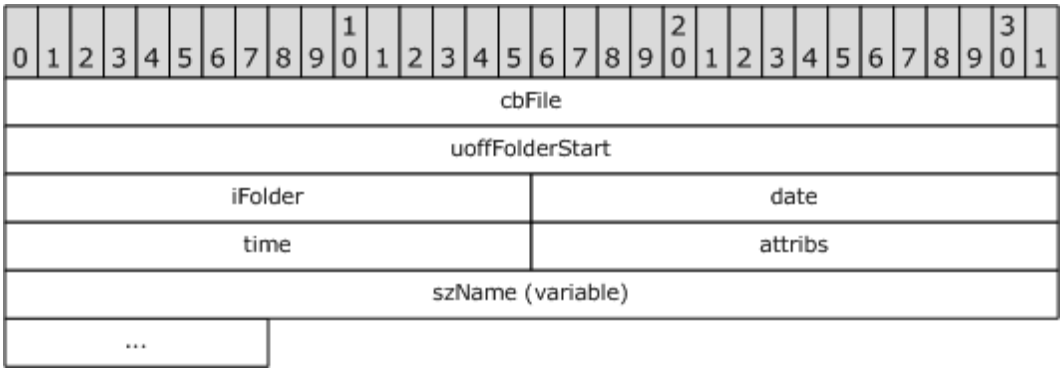

Figure 3: CFFILE structure

2.3.1 u4 cbFile

Specifies the uncompressed size of this file, in bytes.

2.3.2 u4 uoffFolderStart

Specifies the uncompressed offset, in bytes, of the start of this file's data. For the first file in each folder, this value will usually be zero. Subsequent files in the folder will have offsets that are typically the running sum of the cbFile values.

2.3.3 u2 iFolder

Index of the folder containing this file's data. A value of zero indicates that this is the first folder in this cabinet file. The special iFolder values ifoldCONTINUED_FROM_PREV and ifoldCONTINUED_PREV_AND_NEXT indicate that the folder index is actually zero, but that extraction of this file would have to begin with the cabinet named in CFHEADER.szCabinetPrev. The special iFolder values ifoldCONTINUED_PREV_AND_NEXT and ifoldCONTINUED_TO_NEXT indicate that the folder index is actually one less than CFHEADER.cFolders, and that extraction of this file will require continuation to the cabinet named in CFHEADER.szCabinetNext.

```
#define ifoldCONTINUED_FROM_PREV      (0xFFFD)
#define ifoldCONTINUED_TO_NEXT        (0xFFFE)
#define ifoldCONTINUED_PREV_AND_NEXT  (0xFFFF)
```

2.3.4 u2 date

Date of this file, in the format ((year–1980) << 9)+(month << 5)+(day), where month={1..12} and day={1..31}. This "date" is typically considered the "last modified" date in local time, but the actual definition is application-defined.

2.3.5 u2 time

Time of this file, in the format (hour << 11)+(minute << 5)+(seconds/2), where hour={0..23}. This "time" is typically considered the "last modified" time in local time, but the actual definition is application-defined.

2.3.6 u2 attribs

Attributes of this file; can be used in any combination:

```
#define _A_RDONLY      (0x01)   /* file is read-only */
#define _A_HIDDEN      (0x02)   /* file is hidden */
#define _A_SYSTEM      (0x04)   /* file is a system file */
#define _A_ARCH        (0x20)   /* file modified since last backup */
#define _A_EXEC        (0x40)   /* run after extraction */
#define _A_NAME_IS_UTF (0x80)   /* szName[] contains UTF */
```

All other attribute bit values are reserved; they SHOULD be set to 0 and MUST be ignored.

2.3.7 u1 szName[]

The NULL-terminated name of this file. Note that this string can include path separator characters. When attribute **_A_NAME_IS_UTF** is set as specified in section 2.3.6, this string can be converted directly to **Unicode**, avoiding locale-specific dependencies. When attribute **_A_NAME_IS_UTF** is not set, this string is subject to interpretation depending on locale.

When a string containing Unicode characters larger than 0x007F is encoded in the **szName** field, the **_A_NAME_IS_UTF** attribute SHOULD be included in the file's attributes, as specified in section 2.3.6. When no characters larger than 0x007F are in the name, the **_A_NAME_IS_UTF** attribute SHOULD NOT be set. If byte values larger than 0x7F are found in CFFILE.szName, but the **_A_NAME_IS_UTF** attribute is not set, the characters SHOULD be interpreted according to the current locale.

2.4 CFDATA

Each CFDATA record describes some amount of compressed data. The first CFDATA entry for each folder is located using CFFOLDER.coffCabStart. Subsequent CFDATA records for this folder are contiguous.

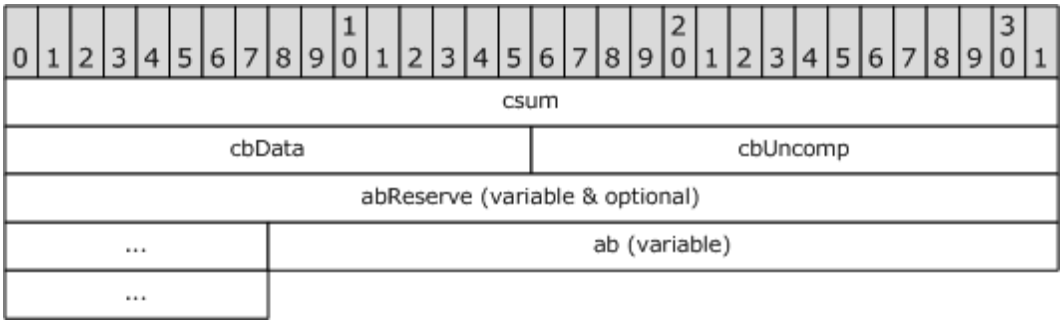

Figure 4: CFDATA structure

2.4.1 u4 csum

Checksum of this **CFDATA** structure, from CFDATA.cbData through CFDATA.ab[cbData-1]. It can be set to zero if the checksum is not supplied.

2.4.2 u2 cbData

Number of bytes of compressed data in this CFDATA record. When **cbUncomp** is zero, this field indicates only the number of bytes that fit into this cabinet file.

2.4.3 u2 cbUncomp

The uncompressed size of the data in this CFDATA entry in bytes. When this CFDATA entry is continued in the next cabinet file, **cbUncomp** will be zero, and **cbUncomp** in the first CFDATA entry in the next cabinet file will report the total uncompressed size of the data from both CFDATA blocks.

2.4.4 u1 abReserve[CFHEADER.cbCFData] (optional)

If CFHEADER.flags.cfhdrRESERVE_PRESENT is set and cbCFData is non-zero, then this field contains per-datablock application information. This field is defined by the application, and is used for application-defined purposes.

2.4.5 u1 ab[cbData]

The compressed data bytes, compressed using the CFFOLDER.typeCompress method. When **cbUncomp** is zero, these data bytes MUST be combined with the data bytes from the next cabinet's first CFDATA entry before decompression.

When CFFOLDER.typeCompress indicates that the data is not compressed, this field contains the uncompressed data bytes. In this case, **cbData** and **cbUncomp** will be equal unless this CFDATA entry crosses a cabinet file boundary.

3 Structure Examples

The following is a simple example of a cabinet file that contains two small text files, stored uncompressed for clarity.

The following is a hexadecimal representation of the cabinet file.

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000   4D 53 43 46 00 00 00 00-FD 00 00 00 00 00 00 00   MSCF
010   2C 00 00 00 00 00 00 00-03 01 01 00 02 00 00 00
020   22 06 00 00 5E 00 00 00-01 00 00 00 4D 00 00 00
030   00 00 00 00 00 00 6C 22-BA 59 20 00 68 65 6C 6C              hell
040   6F 2E 63 00 4A 00 00 00-4D 00 00 00 00 00 6C 22   o.c
050   E7 59 20 00 77 65 6C 63-6F 6D 65 2E 63 00 BD 5A       welcome.c
060   A6 30 97 00 97 00 23 69-6E 63 6C 75 64 65 20 3C         #include <
070   73 74 64 69 6F 2E 68 3E-0D 0A 0D 0A 76 6F 69 64   stdio.h>    void
080   20 6D 61 69 6E 28 76 6F-69 64 29 0D 0A 7B 0D 0A    main(void)  {
090   20 20 20 20 70 72 69 6E-74 66 28 22 48 65 6C 6C       printf("Hell
0A0   6F 2C 20 77 6F 72 6C 64-21 5C 6E 22 29 3B 0D 0A   o, world!\n");
0B0   7D 0D 0A 23 69 6E 63 6C-75 64 65 20 3C 73 74 64   }  #include <std
0C0   69 6F 2E 68 3E 0D 0A 0D-0A 76 6F 69 64 20 6D 61   io.h>    void ma
0D0   69 6E 28 76 6F 69 64 29-0D 0A 7B 0D 0A 20 20 20   in(void)  {
0E0   20 70 72 69 6E 74 66 28-22 57 65 6C 63 6F 6D 65    printf("Welcome
0F0   21 5C 6E 22 29 3B 0D 0A-7D 0D 0A 0D 0A           !\n");  }
```

The following is a structure representation of the cabinet file.

```
00..23      CFHEADER
   00..03   signature = 0x4D, 0x53, 0x43, 0x46
   04..07   reserved1
   08..0B   cbCabinet = 0x000000FD (253)
   0C..0F   reserved2
   10..13   coffFiles = 0x0000002C
   14..17   reserved3
   18..19   versionMinor, Major = 1.3
   1A..1B   cFolders = 1
   1C..1D   cFiles = 2
   1E..1F   flags = 0 (no reserve, no previous or next cabinet)
   20..21   setID = 0x0622
   22..23   iCabinet = 0

24..2B      CFFOLDER[0]
   24..27   coffCabStart = 0x0000005E
   28..29   cCFData = 1
   2A..2B   typeCompress = 0 (none)

2C..43      CFFILE[0]
   2C..2F   cbFile = 0x0000004D (77 bytes)
   30..33   uoffFolderStart = 0x00000000
   34..35   iFolder = 0
   36..37   date = 0x226C = 0010001 0011 01100 = March 12, 1997
   38..39   time = 0x59BA = 01011 001101 11010 = 11:13:52 AM
   3A..3B   attribs = 0x0020 = _A_ARCH
   3C..43   szName = "hello.c" + NUL

44..5D      CFFILE[1]
```

```
   44..47   cbFile = 0x0000004A (74 bytes)
   48..4B   uoffFolderStart = 0x0000004D
   4C..4D   iFolder = 0
   4E..4F   date = 0x226C = 0010001 0011 01100 = March 12, 1997
   50..51   time = 0x59E7 = 01011 001111 00111 = 11:15:14 AM
   52..53   attribs = 0x0020 = _A_ARCH
   54..5D   szName = "welcome.c" + NUL

5E..FD      CFDATA[0]
   5E..61   csum = 0x30A65ABD
   62..63   cbData = 0x0097 (151 bytes)
   64..65   cbUncomp = 0x0097 (151 bytes)
   66..FD   ab[0x0097] = uncompressed file data
```

3.1 Checksum Method

The computation and verification of checksums found in CFDATA entries cabinet files is done by using a function described by the following mathematical notation. When checksums are not supplied by the cabinet file creating application, the checksum field is set to zero. Cabinet extracting applications do not compute or verify the checksum if the field is set to zero.

Given the n-tuple T of bytes

$$\forall n \in \mathbb{N}$$

$$T = [a_1, a_2, \ldots, a_n]$$

And the function S

$$S(b,x) = \begin{cases} 0 & if\ x < 4\ and\ b > n\%4 \\ a_{n-b+1} & if\ x < 4\ and\ b \leq n\%4 \\ a_{n-x+b} \oplus S(b, x-4) & if\ x \geq 4 \end{cases}$$

The Cabinet Checksum of T is defined as the 4-tuple C of bytes

$$C = [S(1,n), S(2,n), S(3,n), S(4,n)]$$

The Cabinet Checksum is a four-byte longitudinal parity check with special handling for remainder bytes, as follows:

- The data is broken into four-byte words.
- If any bytes remain, an additional four-byte word is constructed beginning with the last remainder byte, proceeding in reverse, and padding with one to three null bytes.
- The Cabinet Checksum is the exclusive-or of all these four-byte words.

The checksums for non-split CFDATA blocks are computed first on the compressed data bytes, then on the CFDATA header area, starting at the **CFDATA.cbData** field.

When blocks are split across cabinet file boundaries, the checksum for the partial block at the end of a cabinet file is computed first on the partial field of compressed data bytes, then on the header.

The checksum for the residual block in the next cabinet file is computed first on the remainder of the field of compressed data bytes, then on the header.

4 Security Considerations

None.

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products:

- Microsoft® Office Outlook® 2003
- Microsoft® Exchange Server 2003
- Microsoft® Office Outlook® 2007
- Microsoft® Exchange Server 2007
- Microsoft® Outlook® 2010
- Microsoft® Exchange Server 2010

Exceptions, if any, are noted below. If a service pack number appears with the product version, behavior changed in that service pack. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that product does not follow the prescription.

6 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

7 Index