[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.
03/18/2011	6.0	Major	Significantly changed the technical content.

Table of 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.

Sections 1.7 and 2 of this specification are normative and contain RFC 2119 language. All other sections and examples in this specification are informative.

1.1 Glossary

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

ASCII
little-endian
Unicode

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

cabinet (.cab) file

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-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

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

1.3 Overview

Each file stored in a cabinet is stored completely within a single folder. A **cabinet (.cab) 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.

This specification does not describe compression and decompression schemes.

1.4 Relationship to Protocols and Other Structures

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

1.5 Applicability Statement

This structure is designed to be a container for compressed data.

1.6 Versioning and Localization

The major version of this structure is 1. The minor version is 3.

1.7 Vendor-Extensible Fields

None.

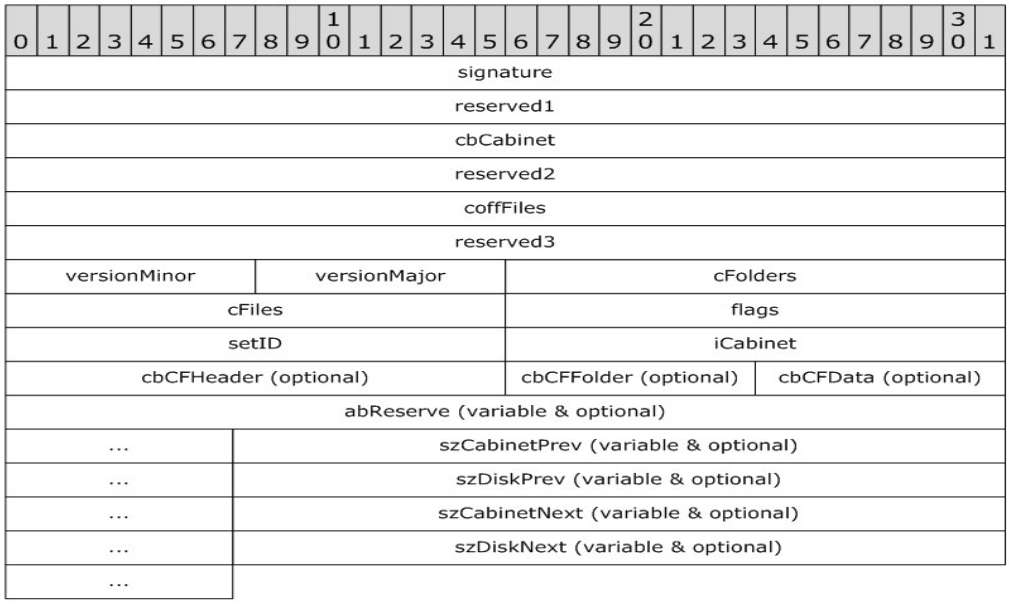
2 Structures

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

The cabinet (.cab) 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 shown in the following packet diagram provides information about this cabinet (.cab) file.



signature (4 bytes): 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 (.cab) file.

reserved1: Reserved field; MUST be set to 0 (zero).

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

reserved2: Reserved field; MUST be set to 0 (zero).

coffFiles: Specifies the absolute file offset, in bytes, of the first **CFFILE** field entry.

reserved3: Reserved field; MUST be set to 0 (zero).

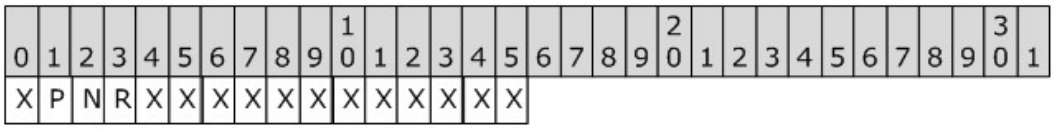
versionMinor: Specifies the minor cabinet file format version. This value MUST be set to 3 (three).

versionMajor: Specifies the major cabinet file format version. This value MUST be set to 1 (one).

cFolders: Specifies the number of **CFFOLDER** field entries in this cabinet file.

cFiles: Specifies the number of **CFFILE** field entries in this cabinet file.

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



P: (cfhdrPREV_CABINET) The flag 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 structure**. The value is 0x0001.

N: (cfhdrNEXT_CABINET) The flag 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 structure**. The value is 0x0002.

R: (cfhdrRESERVE_PRESENT) The flag is set if if this cabinet file contains any reserved fields. When this bit is set, the **cbCFHeader**, **cbCFFolder**, and **cbCFData** fields are present in this **CFHEADER structure**. The value is 0x0004.

X: (Reserved) These bit fields SHOULD be set to 0 and MUST be ignored.

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** field value. 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.

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

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

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

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

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

szCabinetPrev (variable) (optional): If the **flags.cfhdrPREV_CABINET** field is not set, this field is not present. This is a NULL-terminated **ASCII** string that contains 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," the **szCabinetPrev** field would indicate the name of the cabinet to examine.

szDiskPrev (variable) (optional): If the **flags.cfhdrPREV_CABINET** field is not set, then this field is not present. This is a NULL-terminated ASCII string that contains a descriptive name for the media that contains the file named in the **szCabinetPrev** field, 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.

szCabinetNext (variable) (optional): If the **flags.cfhdrNEXT_CABINET** field is not set, this field is not present. This is a NULL-terminated ASCII string that contains the file name of the next cabinet file in a set. The string can contain up to 255 bytes, plus the NULL byte. Files that extend beyond the end of the current cabinet file are continued in the named cabinet file.

szDiskNext (variable) (optional): If the **flags.cfhdrNEXT_CABINET** field is not set, this field is not present. This is a NULL-terminated ASCII string that contains a descriptive name for the media that contains the file named in the **szCabinetNext** field, 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.

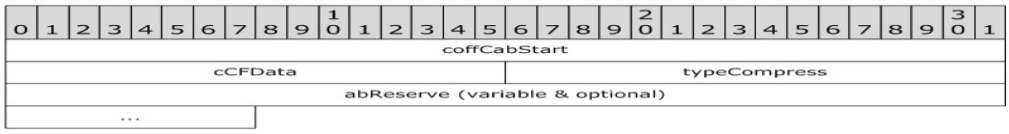
2.2 CFFOLDER

Each **CFFOLDER** structure contains information about one of the folders or partial folders stored in this cabinet file, as shown in the following packet diagram. The first **CFFOLDER** structure entry immediately follows the **CFHEADER** structure entry. The **CFHEADER.cFolders** field indicates how many **CFFOLDER** structure 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 structure entries actually refer to folder fragments, not necessarily complete folders. A **CFFOLDER** structure is the beginning of a folder if the **iFolder** field value in the first file that references 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.



coffCabStart: Specifies the absolute file offset of the first **CFDATA** field block for the folder.

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** structure 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.

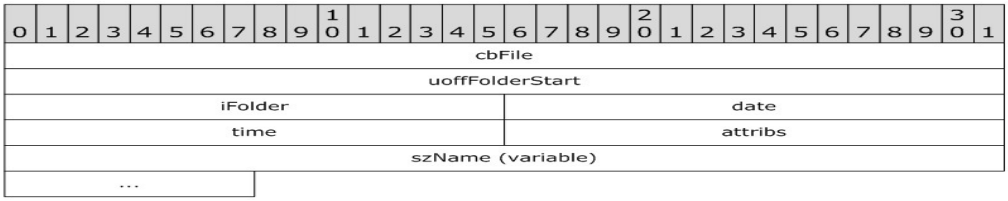
typeCompress: Indicates the compression method used for all **CFDATA** structure entries in this folder. The following are the valid values.

Flag	Description	Value
tcompMASK_TYPE	Mask for compression type.	0x000F
tcompTYPE_NONE	No compression.	0x0000
tcompTYPE_MSZIP	MSZIP compression.	0x0001
tcompTYPE_QUANTUM	Quantum compression.	0x0002
tcompTYPE_LZX	LZX compression.	0x0003

abReserve (variable) (optional): If the **CFHEADER.flags.cfhdrRESERVE_PRESENT** field is set and the **cbCFFolder** field 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** structure contains information about one of the files stored (or at least partially stored) in this cabinet, as shown in the following packet diagram. The first **CFFILE** structure entry in each cabinet is found at the absolute offset **CFHEADER.coffFiles** field. **CFHEADER.cFiles** field indicates how many of these entries are in the cabinet. The **CFFILE** structure entries in a cabinet are ordered by **iFolder** field value, and then by the **uoffFolderStart** field value. Entries for files continued from the previous cabinet will be first, and entries for files continued to the next cabinet will be last.



cbFile: Specifies the uncompressed size of this file, in bytes.

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** field values.

iFolder: Index of the folder that contains this file's data. A value of zero indicates that this is the first folder in this cabinet file. The special **iFolder** field 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 the **CFHEADER.szCabinetPrev** field. The special **iFolder** field values "ifoldCONTINUED_PREV_AND_NEXT" and "ifoldCONTINUED_TO_NEXT" indicate that the folder index is actually one less than THE **CFHEADER.cFolders** field value, and that extraction of this file will require continuation to the cabinet named in the **CFHEADER.szCabinetNext** field.

iFolder field value name	Value
ifoldCONTINUED_FROM_PREV	0xFFFD
ifoldCONTINUED_TO_NEXT	0xFFFE
ifoldCONTINUED_PREV_AND_NEXT	0xFFFF

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.

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.

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

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	R	H	S	X	X	A	E	U	X	X	X	X	X	X	X																

R: (_A_RDONLY) File is read-only.

H: (_A_HIDDEN) File is hidden.

S: (_A_SYSTEM) File is a system file.

A: (_A_ARCH) File has been modified since last backup.

E: (_A_EXEC) File will be run after extraction.

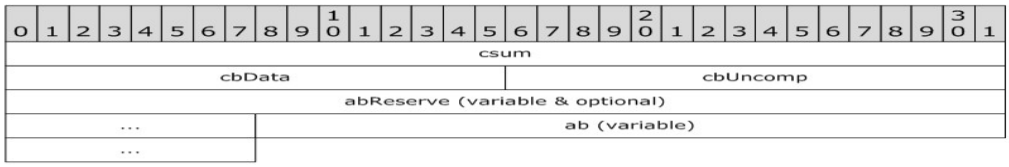
U: (_A_NAME_IS_UTF) The szName field contains UTF.

X: (Reserved) These bit fields SHOULD be set to 0 and MUST be ignored.

szName (variable): The NULL-terminated name of this file. Note that this string can include path separator characters. The string can contain up to 256 bytes, plus the NULL byte. When the **_A_NAME_IS_UTF** attribute is set, this string can be converted directly to **Unicode**, avoiding locale-specific dependencies. When the **_A_NAME_IS_UTF** attribute is not set, this string is subject to interpretation depending on locale. When a string that contains 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. 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** field, but the **_A_NAME_IS_UTF** attribute is not set, the characters SHOULD be interpreted according to the current location.

2.4 CFDATA

Each **CFDATA** structure describes some amount of compressed data, as shown in the following packet diagram. The first **CFDATA** structure entry for each folder is located by using the **CFFOLDER.coffCabStart** field. Subsequent **CFDATA** structure records for this folder are contiguous.



csum: Checksum of this **CFDATA** structure, from the **CFDATA.cbData** through the **CFDATA.ab[cbData-1]** fields. It can be set to 0 (zero) if the checksum is not supplied.

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

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

abReserve (variable) (optional): If the **CFHEADER.flags.cfhdrRESERVE_PRESENT** flag is set and the **cbCFData** field value is non-zero, this field contains per-datablock application information. This field is defined by the application, and it is used for application-defined purposes.

ab (variable): The compressed data bytes, compressed by using the **CFFOLDER.typeCompress** method. When the **cbUncomp** field value is zero, these data bytes MUST be combined with the data bytes from the next cabinet's first **CFDATA** structure entry before decompression. When the **CFFOLDER.typeCompress** field indicates that the data is not compressed, this field contains the uncompressed data bytes. In this case, the **cbData** and **cbUncomp** field values will be equal unless this **CFDATA** structure 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** structure 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 0 (zero). Cabinet extracting applications do not compute or verify the checksum if the field is set to 0 (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** structure 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.

4.1 Security Considerations for Implementers

None.

4.2 Index of Security Fields

None.

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. 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 that is 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 the product does not follow the prescription.

6 Change Tracking

This section identifies changes that were made to the [MS-CAB] protocol document between the November 2010 and March 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1 Introduction	Added information about which sections of the specification are normative and can contain RFC 2119 language.	Y	New content added for template compliance.
1.3 Overview	Added a statement about compression schemes.	N	Content updated.
1.5 Applicability Statement	Added an applicability statement.	N	New content added.
2.1 CFHEADER	Added information about the reserved3 field.	N	Content updated.
2.1 CFHEADER	Added information about the values for the versionMinor and versionMajor fields.	Y	Content updated.
2.1 CFHEADER	Moved field specifications from subsections to section 2.1.	N	Content updated for template compliance.
2.1 CFHEADER	Updated the flags field.	N	Content updated.
2.2 CFFOLDER	Moved field specifications from subsections to section 2.2.	N	Content updated for template compliance.
2.3 CFFILE	Moved field specifications from subsections to section 2.3.	N	Content updated for template compliance.
2.3	Updated the description of the attribs field.	N	Content updated for

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
CFFILE			template compliance.
2.3 CFFILE	Added the upper limit of the szName field.	N	Content updated.
2.4 CFDATA	Moved field specifications from subsections to section 2.4.	N	Content updated for template compliance.
4.1 Security Considerations for Implementers	Added section.	N	New content added for template compliance.
4.2 Index of Security Fields	Added section.	N	New content added for template compliance.
5 Appendix A: Product Behavior	Removed all product versions from the list of applicable products.	Y	Content updated.

7 Index