Apple II
File Type Notes

Developer Technical Support

File Type: $E0 (224)
Auxiliary Type: $0002 & $0003

Full Name: AppleDouble Header File (Auxiliary Type $0002)
AppleDouble Data File (Auxiliary Type $0003)
Short Name: AppleDouble Header (Auxiliary Type $0002)
AppleDouble Data (Auxiliary Type $0003)

Revised by: Matt Deatherage November 1990
Written by: Matt Deatherage March 1989

Files of these types and auxiliary types contain file data in AppleDouble format.
Changes since March 1990: Added information about AppleDouble 2.0.

AppleDouble is one of two standards (the other is AppleSingle) put forth by Apple Computer, Inc. for representing files on foreign file systems while preserving all attributes of the file's home system on file systems that do not support the same attributes.

Experience indicated that a single format would be inadequate to cover all cases. Two closely related formats, however, can serve most needs. Although the primary impetus for developing these formats is storing extended files (files with both resource and data forks) on file systems that do not support the notion of two forks, the proposed formats are general enough that they can be used to represent a file from any file system on any other file system.

AppleDouble keeps the data fork as a separate file from the file attributes and the resource fork, and this Note describes this file format. AppleSingle keeps all attributes and the contents of both forks in a single file in the foreign file system, and is described in the File Type Note for File Type $E0, Auxiliary Type $0001.

AppleSingle is intended to be used primarily as a storage format, especially for cases where you must store an extended file on a foreign file system and later reconstruct the extended file. AppleDouble is more appropriate for applications where the users of the foreign file system might want to modify the contents of the file. Since most applications keep file data in the data fork, AppleDouble format saves the contents of the data fork in one file. All other file attributes, including the resource fork, are kept in a separate file.

Reasons for Using AppleDouble

There are several reasons for supporting an interchange format between file systems. Perhaps the most germane is one of the least obvious: handling extended files on foreign file systems which do not support extended files.

For example, the ProDOS FST in GS/OS can create an extended file on a ProDOS disk. However, ProDOS 8 is unable to operate on the file, since it sees it as having an unsupported storage type. If a telecommunications program or other utility capable of transferring files is operating under ProDOS 8 and attempts to receive an extended file, it is unable to create the file.

At this point, the application could use `READ_BLOCK` and `WRITE_BLOCK` commands, along with a knowledge of the ProDOS file system, to create the file on its own. However, this is strongly **discouraged**. The ProDOS file system format for extended files is not documented and could change in the future. In addition, the program could be running on a eight-bit system. If the disk is only used on an eight-bit system, the extended files would not only be unwanted, but also unremovable without using the disk on an Apple IIGS or later system running GS/OS.

However, if the application is aware of the AppleDouble format, it can quickly store an extended file in AppleDouble, leaving the conversion back to the extended file to GS/OS, or another operating system. This is the recommended way for ProDOS 8 applications to create and handle extended files. Use either AppleSingle or AppleDouble.

AppleDouble Format

AppleDouble consists of two files, an AppleDouble Header File and an AppleDouble Data File. The AppleDouble Header file contains a header followed by data. The header consists of several fixed fields and a list of entry descriptors, each pointing to an entry. Apple defines these standard entries: Resource Fork, Real Name (name in the home file system), Comment, Icon and File Info. Each entry is optional, so it may not appear in the file. We also define the new entry Data Pathname, pointing to the pathname of the AppleDouble Data File. The Header File has exactly the same format as an AppleSingle file, except it has no data fork entry. The AppleDouble Data File consists of just the data fork of the file, with no extra header at all.

Note: All numeric entries, including entries representing ProDOS data structures (such as file type and auxiliary type) are **Reverse** ordered. This is provided so any host CPU can attempt to interpret entries in the header without having to know the standard byte-ordering of the home file system. Therefore, in this Note you see descriptive entries like “Rev. 4 Bytes.” This serves as a reminder that **all** header fields are stored high byte first, even though the notation **Bytes** does not imply any specific ordering in other File Type Notes.

Also note that ASCII strings are not stored in reverse order, just non-ASCII constants.

The Header in the Header File:

Magic Number	**Rev. Long**	The Magic Number field is modeled after the feature in UNIX. It is intended to be used in whatever way the foreign file

		system distinguishes a file as AppleDouble format. See the section "Identifying AppleDouble Files." The Magic Number for AppleDouble format is $00051607, which is stored reverse as $00 $05 $16 $07 (reverse of normal 65816/6502 order).
Version Number	**Rev. Long**	The version of AppleDouble format, in case the format evolves (more fields may be added to the header). The version described here is $00010000, stored (reverse) as $00 $01 $00 $00.

Home File System	**16 Bytes**	A fixed-length, 16-byte ASCII string not preceded by a length byte, but possibly padded with blanks. Apple has defined these values: "ProDOS" $50726F444F532020202020202020202020 "Macintosh" $4D6163696E746F736820202020202020 "MS-DOS" $4D532D444F532020202020202020202020 "Unix" $556E9878202020202020202020202020 "VAX VMS" $56415820564D532020202020202020202020 Apple welcomes suggestions for other file systems that should be included in this list.
Number of entries	**Rev. Word**	Tells how many different entries are included in the file. This unsigned reverse word may be zero. If it is non-zero, then that number of entry descriptors immediately follows this field.

For Each Entry:

Entry ID	**Rev. Long**	Identifies the entry. Apple has defined the following Entry IDs and their values: 1 = Data Fork 2 = Resource Fork 3 = Real Name (The file's name in the home file system) 4 = Comment* (standard Macintosh comment) 5 = Icon, B&W* (standard Macintosh black and white icon) 6 = Icon, Color* (reserved for Macintosh color icon) 7 = File Info (file attributes, dates, etc.) 9 = Finder Info* (standard Macintosh Finder Info) Entry IDs marked with asterisks (*) are not used for most files created under ProDOS or GS/OS. Furthermore, icon entries probably do not appear in most files since they are typically stored as a bundle in the application file's resource fork on the Macintosh. Apple reserves the range of Entry IDs from $0 to $7FFFFFFF for future use. The rest of the range is available for other systems to define their own entries. Apple does not arbitrate the use of the rest of the range. Descriptions of the standard entries are given below.
Offset	**Rev. Long**	An unsigned reverse long which indicates the byte offset from the start of the file to the start of the entry.
Entry Length	**Rev. Long**	An unsigned reverse long which indicates the length of the entry in bytes. The length may be zero.

Standard Entries:

The Real Name Entry:

The Real Name entry indicates the file's original filename in the host file system. This is not a Pascal or C string; it is just ASCII data. The length is indicated by the Entry Length field for the Real Name entry.

The File Info Entry:

The File Info entry (Entry ID = 7) is different for each home file system. For ProDOS files, the entry is 16 bytes long and consists of the creation date and time and the modification date and time in ProDOS 8 (ProDOS 16/class zero GS/OS) form, the access word, a two-byte file type and four-byte auxiliary type. This is detailed in standard format below, along with defined File Info entries for some other file systems.

ProDOS:

Create Date	**Rev. 2 Bytes**	Creation date packed into standard ProDOS 8 format.
Create Time	**Rev. 2 Bytes**	Creation time packed into standard ProDOS 8 format.
Modification Date	**Rev. 2 Bytes**	Modification date packed into standard ProDOS 8 format.
Modification Time	**Rev. 2 Bytes**	Modification time packed into standard ProDOS 8 format.
Access	**Rev. Word**	The file's access. This may be used directly in ProDOS 16 or GS/OS calls; only the low byte is significant to ProDOS 8.
File Type	**Rev. Word**	The file type of the original file. Only the low byte is significant to ProDOS 8.
Auxiliary Type	**Rev. Long**	The auxiliary type of the original file. Only the low word is significant to ProDOS 8.

Note: Although the ProDOS Access field, File Type and Auxiliary Type are the same length as found in ProDOS 16 and GS/OS structures, the Create and Modification Dates and Times are stored in two-byte (albeit byte-reversed) ProDOS 8 format, not eight-byte Apple IIGS format.

Macintosh:

Create Date	**Rev. Long**	Unsigned number of seconds between January 1, 1904, and the creation time of this file.
Modification Date	**Rev. Long**	Unsigned number of seconds between January 1, 1904, and the last modification of this file.
Last Backup Date	**Rev. Long**	Unsigned number of seconds between January 1, 1904, and the last backup time of this file.
Attributes	**Rev. Long**	32 boolean flags. Once the bytes are unreversed, bit zero is the locked bit and bit one is the protected bit.

MS-DOS:

Modification Date	**Rev. 4 Bytes**	MS-DOS format modification date.
Attributes	**Rev. 2 Bytes**	MS-DOS attributes.

Unix:

Create Date/Time	**Rev. 4 Bytes**	Unix creation date and time.
Last Use Date/Time	**Rev. 4 Bytes**	Unix time for the last time this file was used.
Last Mod. Date/Time	**Rev. 4 Bytes**	Unix time for the last time this file was modified.

The Finder Info Entry:

The Finder Info entry (Entry ID = 9) is for files where the host file system is Macintosh. It consists of 16 bytes of Finder Info followed by 16 bytes of Extended Finder Info. These are the

fields `ioFlFndrInfo` followed by `ioFlXFndrInfo`, as described in *Inside Macintosh*, Volume IV-183. Newly created files have zeroes in all Finder Info subfields. If you are creating an AppleDouble file whose home system is Macintosh, you may zero all unknown fields, but you may want to set the `fdType` and `fdCreator` subfields.

The Data Pathname Entry:

The Data Pathname entry (Entry ID = 100) is defined for the first time in this Note. It consists of a class one GS/OS input string noting the pathname of the AppleDouble Data File as originally created:

Path Length	**Rev. Word**	The length of the pathname.
Pathname	**Bytes**	ASCII pathname of the AppleDouble Data File when created.

For strategies on using this segment (or not using it) to find the AppleDouble Data File, see the section “Finding the AppleDouble Data File.”

The Entries in the Header File:

The entries themselves follow the header field and the entry descriptors. The actual data representing each entry must be in a single, contiguous block. The offset field in that entry’s descriptor points to it. The entries could appear in any order, but since the data fork is the entry that is most commonly extended, Apple strongly recommends that the data fork always be kept last in the file to facilitate its extension. Apple also recommends that those entries that are most often read, such as Real Name, File Info (and Finder Info if present) be kept as close as possible to the header to maximize the probability that a read of the first few blocks of the file retrieves these entries.

It is possible to have holes in the file (unused space between entries). To find the holes, you must take the list of entry descriptors and sort them into increasing offset order. If the offset field of an entry is greater than the offset plus the length of the previous entry (sorted), then a hole exists between the entries. You can make use of such holes; for example, if a file’s comment is ten bytes long, you could create a hole of 190 bytes after the comment field to easily allow for the comment to later expand to its maximum length of 200 bytes. Because an AppleDouble file may contain holes, you must find each entry by getting its offset from its entry descriptor, not by assuming that it begins after the previous entry.

Byte ordering in file header fields follows 68000 convention, and each header field has been so noted by the **Reverse** operator.

The AppleDouble Data File

The AppleDouble Data File is simply the data fork of the original file contained in a file of its own. You may create it with a File Type and Auxiliary Type assignment best suited to it, if desired. For example, if the program creating the AppleDouble Data File knows that the data fork contains strictly ASCII text, it could create the file with File Type $04 (Text File) so that other applications can deal with it accordingly.

If the creating program wishes to make no assumptions about the content of the data fork, it is encouraged to create the AppleDouble Data File with file type $E0 and auxiliary type $0003. This identifies the file as an AppleDouble Data File.

Identifying AppleDouble Files

As this is an interchange format, from a ProDOS directory entry there is no way to guarantee which files are AppleDouble files. Apple has allocated File Type $E0, Auxiliary Type $0002 for files which are AppleDouble Header Files, and File Type $E0, Auxiliary Type $0003 for files which are AppleDouble Data Files. We strongly encourage ProDOS 8 and GS/OS applications to use these file type and auxiliary type assignments when creating AppleDouble files.

AppleDouble files which do not have file type $E0 and auxiliary type $0002 or $0003 can most easily be identified by opening them and attempting to interpret them. If it is not an AppleDouble Header File, the Magic Number is not contained in the first four bytes of the file. The chances that the file would begin with those four bytes and **not** be an AppleDouble Header File, on a purely random basis, are 4,294,967,295 to 1. The chances that both the Magic Number and the Version bytes would be the same in a non-AppleSingle file are roughly 1.8 x 10^19 to 1.

Finding the AppleDouble Data File

Since the AppleDouble Data File can be stored anywhere, with any file type and auxiliary type, a program may have to make an effort to find it. We recommend the following steps:

1. If the Data Pathname segment exists, use that pathname. If the path specified in the segment does not exist, extract the file name from the end of the pathname and look in the current directory for that file name.
2. If Step 1 does not find the file (or if the Data Pathname segment does not exist), perform the appropriate Home File System algorithm (described below) to generate the name of the AppleDouble Data File from the AppleDouble Header File.
3. If none of the file names generated in Step 2 are found, ask the user where the AppleDouble Data File is located.

Filename Conventions:

Apple proposes the following standard for identifying AppleDouble Header File names and AppleDouble Data File names from the file's real name.

ProDOS:

To generate the AppleDouble Data File name, use character substitution or deletion to remove illegal characters, and use truncation if necessary to reduce the length of the name to two characters less than the maximum file name length. This would be a maximum of 13, since the maximum file name length is 15.

To generate the AppleDouble Header File name, prefix the AppleDouble Data File name with the characters "R." (uppercase R period).

For example, the file name “This is a Foo File” could translate to an AppleDouble Data File Name of “THIS.IS.A.FOO.” The AppleDouble Header File name would then be “R.THIS.IS.A.FOO.”

Unix:

To generate the AppleDouble Data File name, use character substitution to replace any illegal characters with an underscore (_). Since different Unix systems have different requirements on maximum file name length, do not explicitly truncate the name to a specific length. Rather, allow the truncation to be done by the Unix functions `create()`, `open()`, etc.

To generate the AppleDouble Header File name, A/UX (Apple's implementation of Unix for Macintosh computers) prefixes a percent sign (%) to the AppleDouble Data File name. If necessary, truncate the last character to keep the file name within the legal length range. Other Unix systems may prefix a directory name (e.g., ".AppleDouble/") to the AppleDouble Data File name to create the name of the AppleDouble Header File. In this scheme, all AppleDouble Header Files corresponding to AppleDouble Data files are kept together in a single subdirectory.

MS-DOS:

To generate the AppleDouble Data File name, use character substitution or deletion to remove illegal characters, and use truncation if necessary to reduce the length of the name to eight characters. Then add the MS-DOS extension that is most appropriate to the file (such as "TXT" for a pure text file).

To generate the AppleDouble Header File name, add the extension ".ADF" to the eight-character file name.

In any instance, most programs probably wish to display the names being used for both AppleDouble files, so that the user may keep track of them on disk.

AppleDouble name derivations will be defined for all other file systems of interest. This allows applications running on the foreign file system (and users as well) to see easily which files are AppleDouble pairs. Knowledgeable users, if they know the derivation, could rename or move the files so as to preserve the connection between the two. However, there is no guaranteed way to prevent one file of the pair from being inconsistently renamed, moved, or deleted.

About AppleDouble 2.0

AppleDouble 2.0 is a revision to the original AppleDouble specification described in this Note. AppleDouble 2.0 comes closer to the ideal of an interchange format by allowing file information for multiple file systems in the same AppleDouble file.

AppleDouble 2.0 basically replaces the File Info entry (ID = 7) with a File Dates entry (ID = 8) and one or more host file system entries, such as a Macintosh File Info entry (ID = 10), a ProDOS File Info entry (ID = 11), or an MS-DOS File Info entry (ID = 12). Information on these entries and AppleDouble 2.0 can be found in the *AppleSingle/AppleDouble Formats for Foreign Files Developer's Note*, available from APDA, AppleLink, and the Developer CD series.

Further Reference

- *Inside Macintosh*, Volume IV
- *ProDOS 8 Technical Reference Manual*
- *GS/OS Reference*
- *AppleSingle/AppleDouble Formats for Foreign Files Developer's Note*