Apple II
File Type Notes

Developer Technical Support

File Type: $E0 (224)
Auxiliary Type: $8000

Full Name: Binary II File
Short Name: Binary II File

Written by: Matt Deatherage July 1989

Files of this type and auxiliary type contain other files with their attributes encoded in Binary II format.

Binary II is a widely used and accepted standard for keeping file attributes with files as they are transferred, usually by modem or other form of telecommunication. Files that are known Binary II files should be written to disk with file type $E0 and auxiliary type $8000 as a clear indication to other programs that the file contains files with Binary II specifications.

Binary II was developed by Gary B. Little, author of the Point-To-Point communications product and author of several Apple II reference books. He is also Apple's Product Manager for third-party Development Tools and Languages. Gary welcomes your comments and suggestions on the Binary II standard at the following address:

Gary B. Little
3304 Plateau Drive
Belmont, CA 94002

AppleLink:	LITTLE
AppleLink—Personal Edition:	GaryLittle
CompuServe:	70135,1007
GEnie:	GARY.LITTLE

Why Binary II?

Transferring Apple II files in binary form to commercial information services and bulletin boards (referred to in this Note as "hosts") can be, to put it mildly, a frustrating exercise. Although most hosts are able to receive a file's data in binary form (using protocols such as XMODEM), they don't receive the file's all-important attribute bytes. All the common Apple II file system, notably the ProDOS file system, store the attributes inside the disk directory, not inside the file itself.

The ProDOS attributes are the access code, file type code, auxiliary type code, storage type code, date of creation and last modification, time of creation and last modification, the file size, and the name of the file itself. Under GS/OS, the same parameters exist for other file systems as well as file system-specific information and two-forked file information. It is usually not possible to use a ProDOS file's data without knowing the file's attributes (particularly the file type, auxiliary type, and size). Therefore, ProDOS files uploaded in binary format to a host are useless to those who download them. The same is true for DOS 3.3 and Pascal files.

Many Apple II communication programs use special protocols for transferring file attributes during a binary file transfer, but none of these protocols have been implemented by hosts. These programs are only useful for exchanging files with another Apple II running the same program.

Without a standard like Binary II, the only acceptable way to transfer an Apple II file to a host is to convert it into ASCII text before sending it. Such a text file would contain a listing of an AppleSoft program, or a series of Apple II monitor commands (e.g., `300:A4 32`). Someone downloading a file can convert it to binary form using the AppleSoft `EXEC` command.

The main disadvantage of this technique is that the text version of the file is over three times the size of the original binary file, making it expensive (in terms of time and money) to upload and download. It is also awkward, and sometimes impossible, to perform the binary-to-text or text-to-binary conversion.

The solution to the problem is to upload an encoded binary file which contains not just the file's data, but the file's attributes as well. Someone downloading such a file can then use a conversion program to strip the attributes from the file and create a file with the required attributes.

This Note describes such a format: Binary II. The description of the format is detailed for the purpose of allowing software developers to implement it in Apple II communication programs.

What Binary II is Not

Binary II is **not** an archival or compression standard. It is designed to be a simple method to keep the attributes normally in a disk file's directory entry with the file as it is transferred. Although multiple files may be placed together with Binary II, this is a matter of convenience for telecommunication programs.

A true archival standard must be designed as such, with the capability to manipulate files within the archive as well as linking them together (compressed or uncompressed) for transfer. NuFX (documented in Apple II File Type Note for File Type $E0, Auxiliary Type $8002) is a good example of a robust, full-featured Apple II archival standard.

Binary II is primarily designed to be added to and subtracted from files "on-the-fly" by telecommunication programs. Binary II files should only be found on disks when they are transferred by a telecommunication program that does not have Binary II capabilities, in which case a separate utility (such as Binary Library Utility by Floyd Zink, Jr.) must be used to extract

the files. Telecommunication programs should be able to transfer files without Binary II processing, however, they should support Binary II processing as a **default**.

The Binary II File Format

The Binary II form of a standard file consists of a 128-byte file information header followed by the file's data. The data portion of the file is padded with nulls ($00 bytes), if necessary, to ensure the data length is an even multiple of 128 bytes.

The file information header contains four ID bytes, the attributes of the file (in ProDOS 8 form), and some control information.

The structure of the header is as follows:

+000	ID Bytes	**3 Bytes**	These three bytes are always $0A $47 $4C for identification purposes, so programs may recognize Binary II files as they are received.
+003	Access Code	**Byte**	ProDOS 8 access byte.
+004	File Type	**Byte**	ProDOS 8 file type.
+005	Aux Type	**Word**	ProDOS 8 auxiliary type.
+007	Storage Type	**Byte**	ProDOS 8 storage type value.
+008	File Size	**Word**	The size of the file in 512-byte blocks.
+010	Mod. Date	**2 Bytes**	Date of modification, in ProDOS 8 compressed format.
+012	Mod. Time	**2 Bytes**	Time of modification, in ProDOS 8 compressed format.
+014	Create Date	**2 Bytes**	Date of creation, in ProDOS 8 compressed format.
+016	Create Time	**2 Bytes**	Time of creation, in ProDOS 8 compressed format.
+018	ID Byte	**Byte**	A fourth ID byte. This must always be $02.
+019	Reserved	**Byte**	Reserved, must be set to zero.
+020	EOF	**3 Bytes**	The end-of-file value for the file (low byte first).
+023	File Name	**String**	Pascal string containing the ASCII filename or partial pathname of this file in ProDOS 8 format. The string cannot be longer than 64 characters.

If the File Name **String** is a filename and not a partial pathname, then the following optional parameter may be supplied:

+039	Native Name	**String**	Pascal string containing the ASCII value of the native filename. This string may not be longer than 48 characters, and will not be present if the length byte of File Name (+023) is larger than 15 ($0F). If this field is specified, the File Name field must contain a filename, not a partial pathname.
+088	Reserved	**21 Bytes**	Reserved. These bytes must be set to zero for future compatibility.
+109	GAux Type	**Word**	The high word of the file's GS/OS auxiliary type.
+111	GAccess	**Byte**	The high byte of the file's GS/OS access word.
+112	GFile Type	**Byte**	The high byte of the file's GS/OS file type.
+113	GStorage	**Byte**	The high byte of the file's GS/OS storage type.
+114	GFile Size	**Word**	The high word of the GS/OS file's size in 512-byte blocks.
+116	GEOF	**Byte**	The high byte of the file's GS/OS EOF value.
+117	Disk Space	**Long**	The number of 512-byte disk blocks the files inside the Binary II file will occupy after they've been removed from the Binary II file. (The format of a Binary II file containing multiple files is described later in this Note.) If the number is zero, the creator of the Binary II file didn't bother to calculate

the space needed. This value must be placed in the file information header for the first file inside the Binary II file; it can be set to zero in subsequent headers. A downloading program can inspect Disk Space and abort the transfer immediately if there isn't enough free space on the disk.

+121 OS Type **Byte**

This value indicates the native operating system of the file:

$00	ProDOS or SOS
$01	DOS 3.3
$02	Reserved
$03	DOS 3.2 or DOS 3.1
$04	Apple II Pascal
$05	Macintosh MFS
$06	Macintosh HFS
$07	Lisa Filing System
$08	Apple CP/M
$09	Reserved (returned by the GS/OS Character FST)
$0A	MS-DOS
$0B	High Sierra (CD-ROM)
$0C	ISO 9660 (CD-ROM)
$0D	AppleShare

Note this list is slightly different (in the first three entries) from the standard GS/OS file system ID list. A GS/OS communication program should not place a zero in this field unless the file's native file system truly is ProDOS. The file's native file system is returned in the `file_sys_id` parameter from the `GetDirEntry` call.

+122 Native File Type **Word**

This has meaning only if OS Type is non-zero. If so, it is set to the actual file type code assigned to the file by it's native operating system. (Some operating systems, such as MS-DOS and CP/M, do not use file type codes, however.) Contrast this with the File Type at +004, which is the closest equivalent ProDOS file type. The Native File Type is needed to distinguish files which have the same ProDOS file type, but which may have different file types in their native operating system. Note that if the file type code is only one byte long (the usual case), the high-order byte of Native File Type is set to zero.

+124 Phantom File Flag **Byte**

This byte indicates whether a receiver of the Binary II file should save the file which follows (flag is zero) or ignore it (flag is non-zero). It is anticipated that some communication programs will use phantom files to pass non-essential explanatory notes or encoded information which would be understood only by a receiver using the same communication program. Such programs must not rely on receiving a phantom file, however, since this would mean they couldn't handle Binary II files created by other communication programs. Phantom Files may also be used to pass extended file attributes when available.

			The first two bytes in a phantom file must contain an ID code unique to the communication program, or a universal identifier concerning the contents of the phantom file. Developers must obtain ID codes from Gary Little to ensure uniqueness (see the beginning of this Note for his address). Here is a current list of approved ID codes for phantom files used by Apple II communication programs: $00 $00 ASCII text terminated with a zero byte. $00 $01 Point-to-Point $00 $02 Tele-Master Communications System $00 $03 ProTERM $00 $04 Modem MGR $00 $05 CommWorks $00 $06 MouseTalk $01 $00 `Option_list` data (see later in this Note). The ID bytes are the first two bytes of the phantom file.
+125	Data Flags	**Flag Byte**	Bit 7: 1 = file is compressed. Bit 6: 1 = file is encrypted Bits 5-1: Reserved Bit 0: 1 = file is sparse A Binary II downloading program can examine this byte and warn the user that the file must be expanded, decrypted or unpacked. The person uploading a Binary II file may use any convenient method for compressing, encrypting, or packing the file but is responsible for providing instructions on how to restore the file to its original state.
+126	Version	**Byte**	This release of Binary II has a version number of $01.
+127	Number of Files to Follow	**Byte**	An appealing feature of Binary II is that a single Binary II file can hold multiple disk files, making it easy to keep a group of related files "glued" together when they're sent to a host. This byte contains the number of files in this Binary II file that are behind it. If this is the first file in a Binary II file containing three disk files, this byte would be $02. The second disk file in the same Binary II file would have a value of $01 in this parameter, and the last would have value $00. This count tells the Binary II downloading program how many files are remaining. If any phantom files are included, they must be included in this count.

Filenames and Partial Pathnames

You can put a standard ProDOS filename or a partial pathname in the file information header (but never a complete pathname). Don't use a partial pathname unless you've included, earlier in the Binary II file, file information headers for each of the directories referred to in the partial pathname. Such a header must have its "end of file position" bytes set to zero, and no data blocks for the subdirectory file must follow it.

For example, if you want to send a file whose partial pathname is HELP/GS/READ.ME, first send a file information header defining the HELP/ subdirectory, then one defining the HELP/GS/ subdirectory. If you don't, someone downloading the Binary II file won't be able to convert it because the necessary subdirectories will not exist.

Note: GS/OS communication programs must use the slash (/) as the pathname's separator in any partial pathname it puts in the header. Since GS/OS's standard separator is the colon (:), a conversion may be necessary.

Filename Convention

Whenever a file is sent to a host, the host asks the sender to provide a name for it. If it's a file in Binary II form, the name provided should end in .BNY so its special form will be apparent to anyone viewing a list of filenames. If the file is compacted (using the public-domain Squeeze algorithm) before being converted to Binary II form, use a .BQY suffix instead. If the file is a NuFX archive, use the suffix .BXY.

Identifying Binary II Files

You can determine, while transferring, if a file is in Binary II form by examining the ID bytes at offsets +000, +001, +002 and +018 from the beginning of the file. They must be $0A, $47, $4C and $02, respectively.

Once Binary II files are identified, you can use the data in the file information header to create and open a ProDOS file with the correct name and attributes, transfer the file data in the Binary II file to the ProDOS file, set the ProDOS file size, then close the ProDOS file. You would repeat this for each file contained inside the Binary II file.

Note: The number of 128-byte blocks following the file information header must be derived from the EOF attribute for the file. Calculate the number by dividing the EOF by 128 and adding one to the result if EOF is not 0 or an exact multiple of 128. However, if the file information header defines a subdirectory (the file type is $0F), simple create the subdirectory file. Do not open it and do not try to set its size.

Ideally, all this conversion work will be done automatically by a communication program during file transfer. If not, a separate conversion program (such as the previously mentioned Binary Library Utility, or BLU) must be used to do this for you.

Option_List Phantom Files

GS/OS will return, when asked, an `option_list` for files on many file calls. The `option_list` consists of a Word buffer length (which must be at least $2E), followed by a Word number of bytes GS/OS put in the buffer, a Word GS/OS file system identification, and the given number of bytes of FST-specific information (minus two; the count GS/OS returns includes the file system identifier).

`Option_list` values are FST specific and contain values important to the native file system but not important to GS/OS. For AppleShare, the `option_list` contains Finder Information, parent directory identification, and access privileges. This information should be transferred with files.

Binary II uses a phantom file with identifier $01 $00 to indicate an `option_list`. When this phantom file is seen, the contents should be used as the `option_list` for the file that immediately **follows** this file in the Binary II file. The other attributes of the phantom file must be set to the same values as those for the file immediately following (the file for which the phantom file contains the `option_list`). The EOF for the phantom file must be the size of the `option_list` + 2, and the file size must be adjusted accordingly to account for the phantom file ID bytes.

When receiving a Binary II file, the contents of this phantom file should be used as `option_list` input on a GS/OS `SetFileInfo` call.

If the GS/OS `option_list` returns a total of two bytes (just the `file_sys_ID`), there is no FST-specific information, and the `option_list` phantom file may safely be omitted.

The format of the `option_list` phantom file is as follows:

+000	Phantom ID	**2 Bytes**	The identifying bytes $01 $00.
+002	List Size	**Word**	The length of the bytes in the `option_list`, starting with the file system ID (the next word).
+004	FileSysID	**Word**	A GS/OS (not Binary II) `file_sys_ID` for the volume on which the file was stored.
+006	List Bytes	**Bytes**	The bytes of the option list. There should be (List Size) of them, counting the previous word (FileSysID).

Extended File Considerations

Extended files contain two logical segments: a data fork and a resource fork. These files can be created and manipulated by GS/OS, but not by ProDOS 8 or any other Apple II operating system.

When a GS/OS-based communication program sends an extended file, it must send it in the AppleSingle file format, preceded by a Binary II file information header. (Such a program could easily convert an extended file to AppleSingle format on the fly.) The Binary II header must contain the attributes of the AppleSingle file (including a file type of $E0 and an auxiliary type of $0001) and the "storage type code" field must be $01. (The EOF positions for the data fork and resource fork of the extended file appear in an entry in the AppleSingle file header, not in the Binary II header.)

The AppleSingle format is described in Apple II File Type Note for File Type $E0, Auxiliary Type $0001.

A GS/OS-based communication program that receives an AppleSingle file can easily convert it on the fly to the extended file it defines. ProDOS 8-based communication programs can only save the AppleSingle file to disk because it's not possible (nor is it encouraged to attempt) to create extended files with ProDOS 8 (without using block-level calls); a GS/OS based utility program is needed to convert the AppleSingle file to its extended form.

DOS 3.3 Considerations

With a little extra effort, you can also convert DOS 3.3 files to Binary II form. This involves translating the DOS 3.3 file attributes to the corresponding ProDOS attributes so that you can build a proper file information header.

- Set the name to one that adheres to the stricter ProDOS naming rules and put its length at +023 and the name itself at +024 to +038. Note that the name must be a simple filename and not a pathname. The actual DOS 3.3 filename must be placed at +039 (length) and +040 to +087 (name). (DOS 3.3 actually restricts filenames to 30 characters.)

- Set the ProDOS file type, auxiliary type and access to values which correspond to the DOS 3.3 file type:

DOS 3.3 File Type	ProDOS File Type	ProDOS Auxiliary Type	ProDOS Access
$00 (T)	$04	$0000	$E3
$80 (*T)	$04	$0000	$21
$01 (I)	$FA	$0C00	$E3
$81 (*I)	$FA	$0C00	$21
$02 (A)	$FC	*	$E3
$82 (*A)	$FC	*	$21
$04 (B)	$06	**	$E3
$84 (*B)	$06	**	$21
$08 (S)	$06	$0000	$E3
$88 (*S)	$06	$0000	$21
$10 (R)	$FE	$0000	$E3
$90 (*R)	$FE	$0000	$21
$20 (A)	$06	$0000	$E3
$A0 (*A)	$06	$0000	$E3
$40 (B)	$06	$0000	$E3
$C0 (*B)	$06	$0000	$21

* Set the auxiliary type for an A file to the memory address from which the program was saved. This is usually $0801.

** Set the auxiliary type for a B file to the value stored in the first two bytes of the the file (this is the default load address).

- Set the storage type code to $01.
- Set the size of file in blocks, date of creation, date of modification, time of creation and time of modification all to $0000.
- Set the end-of-file position to the length of the DOS 3.3 file, in bytes. For a B file (code $04 or $84), this number is stored in the third and fourth bytes of the file. For an I file (code $01 or $81) or an A file (code $02 or $82), this number is stored in the first and second bytes of the file.
- Set the operating system type to $01.
- Set the native file type code to the value of the DOS 3.3 file type code.

Attribute bytes inside a DOS 3.3 file (if any) must not be included in the data portion of the Binary II file. This includes the first four bytes of a B (Binary) file, and the first two bytes of an A (AppleSoft) or I (Integer BASIC) file.

Further Reference

- *GS/OS Reference*
- *ProDOS 8 Technical Reference Manual*
- Apple II File Type Note, File Type $E0, Auxiliary Type $0001
- Apple II File Type Note, File Type $E0, Auxiliary Type $8002

- Apple II Miscellaneous Technical Note #14, Guidelines for Telecommunication Programs